MOTION PLANNING FOR REDUNDANT MANIPULATORS AND OTHER HIGH DEGREE-OF-FREEDOM SYSTEMS

A Thesis
Presented to
The Academic Faculty

by

Leo Keselman

In Partial Fulfillment
of the Requirements for the Degree
Master of Science in the
School of Electrical and Computer Engineering

Georgia Institute of Technology
May 2014

MOTION PLANNING FOR REDUNDANT MANIPULATORS AND OTHER HIGH DEGREE-OF-FREEDOM SYSTEMS

Approved by:

Professor Patricio Vela, Advisor
School of Electrical and Computer Engineering
Georgia Institute of Technology

Professor Ayanna Howard
School of Electrical and Computer Engineering
Georgia Institute of Technology

Professor Erik Verriest
School of Electrical and Computer Engineering
Georgia Institute of Technology

Date Approved: April 4, 2014

PREFACE

This dissertation presents original work done by the author between May 2013 and March 2014. Most of the work was done at Georgia Tech's IVALAB under the tutelage of Patricio Vela. The Forage RRT, chapters III and IV of the thesis, is to be published and presented at the 2014 ICRA conference in Hong Kong, China.

The animal foraging literature review was conducted by Patricio Vela. All other work presented was researched, designed, implemented, and tested by the author Leo Keselman aside from the software libraries employed which are cited as necessary.

ACKNOWLEDGEMENTS

Special thanks goes to my advisor, Patricio Vela, without whose patient and challenging guidance, neither this thesis nor the ideas expressed therein may have come to be.

My lab mates also deserve recognition for some advice regarding the thesis topic and many more hours of delightful conversation and friendly ribbing. Without these, the thesis could have been done much quicker but would have been a lot less fun.

Finally, I would like to thank Jennifer, Jezebel, and Chewie for providing comfort when (and this was quite often) software was not working as it should.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

This thesis explores motion planning methods for redundant manipulators. This set of manipulators, having more free axes than the number of degrees of freedom of the space in which they operate, poses a special challenge to motion planning because typical motion planning algorithms would require inverse kinematics to first be calculated on the goal query. This is difficult for redundant manipulators because typically no closed form solution exists and there can be an infinite number of solutions.

Although numerical inverse kinematics algorithms can be used, the case is made that this approach can not guarantee completeness. This is because there is no way to ensure that the goal configuration will be reachable from start without knowing the path to do so. Instead, the focus is turned to a class of algorithms which do not require inverse kinematics, relying instead on the Jacobian to make progress toward the goal. These types of algorithms can be complete because they do not rely on a single or multiple inverse kinematics solutions but rather return the solution which is naturally reached first.

Despite the desirable property of potential completeness, these methods often have trouble navigating cluttered environments. This is because a main tool for effectiveness in this regard, bidirectional approaches, have to be thrown out due to the lack of knowledge of a goal configuration. To overcome this problem, the Forage RRT is proposed which is an RRT(Rapidly-exploring Random Tree) based motion planner which combines a coarse exploratory search tree with many fine goal-directed search trees to provide an effective means of getting around obstacles even with a single-directional approach. The Forage RRT is shown to be able to solve planning queries

much more quickly and reliably than other thus-far proposed methods which do not rely on inverse kinematics in three test cases. Moreover, a learning approach to the Forage RRT and a parallel implementation provide further significant improvements in planning time.

In order to incorporate the Forage RRT into the planning community and allow for easy access to it, it is added to the Open Motion Planning Library. This requires a new state space concept which is tailored to manipulators by allowing sampling, equality, distance, and interpolation calculations to take place regardless of whether the joint configuration of a particular state is known.

The value of the Forage RRT is established by its completeness and relative efficacy on difficult planning problems but an experiment is undertaken to compare the Forage RRT through MoveIt to a number of inverse kinematics/sampling based approaches on several practical planning scenes and queries. The results of the experiment indicate that despite its promise of completeness, the Forage RRT is outperformed quite handily by IK based algorithms such as BKPIECE, RRT Connect, and PRM. The conclusion is drawn that the academic promise of completeness significantly increases the average planning time on the same queries and that if available, inverse kinematics should be used for motion planning to achieve optimal timing results.

CHAPTER I

INTRODUCTION

One of the ultimate goals of motion planning algorithms for manipulators is to allow the manipulator to automatically find a collision-free path in a general environment between any given start joint configuration and a goal point specified as an end-effector configuration. The goal is ideally specified in end-effector space because it is almost always the end-effector which interacts with the object to be manipulated. The specific configuration of the other links of the manipulator is usually not important as long as it satisfies workspace constraints such as avoiding obstacle collisions and respecting joint limits while achieving the desired end-effector configuration. Examples of end-effector tools which interact with objects include hands, magnets, suction cups, paint sprayers, and welders. Thus, any practical manipulator planner has to convert end-effector space goal coordinates to manipulator joint space goal coordinates, i.e. the configuration of the manipulator joints which achieves the given end-effector goal coordinate.

The goal of this thesis is to achieve the fastest algorithm that solves the manipulator motion planning problem wherein the goal is given only as an end-effector configuration. Desirable characteristics for the planner are listed as follows:

1. Complete - a solution path is found given that it exists.

2. General - a manipulator with arbitrary number of links and geometry should be able to use it.

3. Fast - the planner should be able to be used in situations where the environment

may change over time. The planner should be fast enough to be used in a single-query manner, not assuming the environment is static between runs.

4. Smooth - the path returned by the planner should be reasonably short and smooth to reduce energy consumption, wear and tear on the manipulator, and time consumption of execution of the path. Some concessions in this area may be necessary to accommodate the more important goal of fast planning time, but only to a subjectively reasonable level.

Approaches which do not satisfy the desired completeness characteristic are investigated as well to get an understanding about the value of such algorithms and the value of different notions of completeness. In other words, the following questions are also investigated:

1. Are algorithms which aren't strictly complete generally faster?
2. How common are cases where incomplete algorithms fail?
3. Is it possible to overcome such cases in a satisfactory manner using methods outside the planner?
4. How valuable are notions of completeness, especially resolution completeness?

1.1 Existing Approaches

In the literature, the two main approaches to solving the problem of end-effector space goals have been to either figure out the joint space goal coordinate directly using inverse kinematics or to incorporate the search for the joint space goal coordinate into the planning. The two approaches and their inherent benefits and problems are discussed below. This thesis explores and compares both approaches.

1.1.1 Inverse Kinematics

The concept behind this approach is that if it is possible to express the goal as a joint space coordinate, an existing fast planner, such as the RRT, can be used to connect the start to the goal. Unfortunately, it is not possible to express end-effector space coordinates in manipulator joint space coordinates uniquely for a general manipulator. The reason is that, except for certain special geometries, a given end effector configuration can be reached by many different manipulator joint configurations. For redundant manipulators (number of degrees of freedom of the manipulator exceeds the number of degrees of freedom of the rigid body Cartesian space in which it operates), which are the most useful for environments with obstacles, there are an infinite number of solutions to the inverse kinematics problem. There are many approaches which try solve this problem; however, to date, there are no inverse kinematics algorithms for general n-link manipulators which are complete, fast, and return a configuration guaranteed to be reachable from the start configuration.

Moreover, even if a joint space configuration could always be found which was guaranteed to be reachable from the initial configurations, there would be no guarantee that this joint-space configuration is the easiest to reach from the set of all possible joint space solutions to the end-effector configuration specified goal.

This thesis investigates the existing methods of inverse kinematics, especially for redundant manipulators, and tries to understand how much of a hindrance the issues mentioned are to the ultimate goal of a valid motion plan.

1.1.2 Hybrid Approach

The contending approach to the problem described above is to incorporate the search for inverse kinematics into the planning algorithm. Although it sounds counter-intuitive to start search before you know what you're searching for, this can be an effective approach if you know how to take a step towards the goal (Jacobian Inverse

control), or if you can bias your search in the right direction by evaluating the fitness of already explored areas. The benefits of this approach are that, when implemented properly, the planning algorithm is resolution complete and fast. The downside of this approach, to date, is that it struggles in certain situations where the goal is largely occluded by obstacles. This condition makes it difficult for a sampling approach to reach the goal because it requires some low probability conditions to be met. The inverse kinematics approach can solve this problem by being able to simultaneously search backwards from the goal, thus freeing the search from the low probability conditions. This is possible because if the joint space configuration of the goal is known, the goal can be the root for a joint space search tree. This is not the case if the goal joint space configuration is not explicitly known.

In this thesis is described a new approach which helps to alleviate this problem by exploring the space fully for promising approach directions to the goal and only then attempting to actually connect to it. This approach is called the Forage RRT because of its relation to animal foraging behaviors.

CHAPTER II

PREVIOUS WORK

2.1 Motion Planning for Manipulators

2.1.1 Artificial Potential Fields

Achieving tractable, complete motion planning for high dimensional systems such as (redundant) manipulators in general work environments presents several challenges. Algorithms to resolve the high-dimensional aspect have been the first to arise. One of the most famous and most cited approaches is the Artificial Potential Field Method originally presented in [20]. This method created artificial fields in the workspace which tend to produce forces repellent from obstacles but attractive to the goal. In the context of manipulator motion planning, these forces would be applied to the end-effector which, when paired with pseudo-inverse Jacobian control [21], produce fast and effective motion plans, especially locally. Although this method is fast enough to be used in a single query planner, it depends on obstacles being of a simple geometry and suffers from several fundamental issues such as getting stuck in local minima, no passage between closely spaced obstacles, and oscillations in certain conditions [22]. Attempts have been made to solve these issues by the formulation of new potential functions [10, 13]; however, these functions either take prohibitively long to compute or do not resolve all of the aforementioned pitfalls.

2.1.2 Sampling Algorithms

Ariadne's Clew Algorithm [4] was among the first to approximate the feasible joint space via sampling. The main idea behind this algorithm was to implement an explore-search architecture wherein the explore phase would put a landmark in the configuration space, preferably as random as possible and the search phase would

attempt to connect the most recently added landmark to the goal with a simple approach. ACA used this approach to achieve resolution completeness and was shown to be respectably fast for most problems. Algorithms based on probabilistic roadmaps, stemming from [2, 19], use sampling to generate a roadmap for multi-query problems which allows subsequent path planning problems to be solved efficiently. The roadmap is essentially a graph over the configuration space with various methods for constructing the graph performed as a pre-processing step. Planning queries are undertaken by adding the goal as a node in the graph and then performing a search to find the path using an algorithm such as A*. Because the roadmap pre-processing step has a baseline overhead, it is not optimal for our stated goal of a single query planner for general environments.

Within the category of efficient resolution-complete sampling based algorithms, the Rapidly-Exploring Random Tree (RRT) family is popular and effective at solving many high dimensional planning problems [23, 24]. The main strategy in this approach is to grow a tree rooted at the start configuration for the plan and attempt to add the goal as a node into this tree. Once the goal has been added, the desirable property of trees stating that each node has only one parent is exploited to find the planned path quickly be tracing through the parents of the goal node to the initial node. Efficient growth of the tree is achieved by extending the tree in the direction of a random point from its nearest neighbor already in the tree. The algorithm is made practical by adding a goal bias such that steps from the tree are sometimes taken toward the goal instead of toward a random point. For manipulation, standard RRTs depend on start and goal states given as joint space configurations. A bidirectional version of the RRT [24] grows two trees (sometimes randomly, sometimes toward each other), one starting from the goal configuration and the other starting from the initial configuration. This formulation is especially useful when either the start or goal configurations are largely occluded by an obstacle. By growing two trees,

the bidirectional approach allows one tree to escape the occlusion and the other to connect to it. Since goals are most often not specified in joint space for manipulation problems, a method for finding inverse kinematics to transform the goal state into a joint space coordinate is first needed to take advantage of the bidirectional RRT.

2.1.3 Inverse Kinematics

There are several classic and popular inverse kinematics algorithms [8, 14, 15, 26]. These employ variations of numerical methods especially involving the Jacobian to solve the problem. None of these inversion algorithms are complete and guaranteed to be reachable from the start configuration. An inverse kinematics approach that was guaranteed to return a configuration reachable from start was presented in [1]. While it was framed as purely an inverse kinematics algorithm, this approach was really a path planner as the information gathered from the process could be used to find a path from start to goal. The approach presented in this thesis is compared to the one in [1] in Section 3.5.

2.1.4 Incorporating Inverse Kinematics into Planning

When the planning problem also incorporates the inverse kinematics problem as part of it, then the inverse kinematics solution will be resolution complete and guaranteed reachable from the start. In [3], the RRT search is biased to be around the existing nodes in the tree which are (by chance) closest in end-effector space to goal. Significant speed improvements to the biased search were achieved by using the Jacobian transpose or Jacobian pseudo-inverse to take steps in the direction of the goal from existing nodes [33, 34]. These algorithms allow the RRT to have a goal bias even while not knowing the joint-space configuration of the goal. Although these methods are complete and faster than a purely random RRT, they are susceptible to the same problems as a single-directional goal-biased RRT such as the tendency to get stuck when the goal is occluded by an obstacle. Another recent RRT modification with the

same goal is to include growing an additional end-effector space tree which is then used to guide the growth of the joint space tree [12, 35]. The former, [12], grows the end-effector space tree from the goal in a bidirectional-type approach whereas the latter, [35], creates end-effector paths from start to goal. Both approaches struggle to find solutions quickly in bug-trap problems.

CHAPTER III

FORAGE RRT

3.1 Introduction

The Forage RRT is a motion planner developed here based on the concept described in Section 2.1.4. The literature review showed that the Rapidly-exploring Random Tree (RRT) algorithm was the best available option for single-query high degree-of-freedom motion planning due to its efficiency and probabilistic completeness. Previous algorithms have incorporated the search for inverse kinematics into RRTs in several ways. However, none of these algorithms solved the bug trap problem satisfactorily. In this type of problem, the goal is largely occluded by obstacles. The single style RRT algorithm struggles with this type of problem. The bidirectional RRT solves this problem but it is not possible to employ in this case because no manipulator joint configuration is known for the goal. The algorithm proposed here follows a Foraging approach which simultaneously explores the workspace and investigates promising areas for connections to the goal. The exploring takes the form of a large step-size RRT whereas attempting to connect to goal is done with a fine step-size RRT. It is shown that this approach outperforms the other proposed algorithms which tackle problems wherein the goal is given as an end-effector configuration and which do not use any direct inverse kinematics.

3.2 J+RRT[33]

The J+RRT is an important building block for the Forage RRT so it is briefly described in this section. The RRT-JT algorithm [34] describes a modification to the standard goal-biased RRT wherein goal-biased steps are taken using the Jacobian

transpose. It is shown that while the Jacobian transpose does not take the end-effector directly in the direction of the goal, it never takes the end-effector further from the goal. This makes it useful for biasing the growth of the RRT in the direction of the goal without computing any matrix inverses.

More explicitly, the RRT-JT makes the following modifications to the RRT:

1. New nodes are added to the tree not only as a manipulator joint configuration, as usual, but also as a corresponding end-effector configuration in Cartesian space.

2. Steps to goal are taken from the goal's nearest neighbor in the RRT by the Cartesian distance metric. New manipulator joint configurations are added to the RRT according to

$$\alpha_{new} = \alpha_{near} + J^T(\alpha_{near}) * \frac{x_{near} - x_{goal}}{\|x_{near} - x_{goal}\|} * \epsilon$$

   where α is the manipulator joint configuration, x is the Cartesian coordinate, and ϵ is the step size of the RRT.

The J+RRT modifies the RRT-JT by using the Jacobian pseudo-inverse rather than the Jacobian transpose in goal-directed steps. More explicitly, goal-directed steps are taken according to

$$\alpha_{new} = \alpha_{near} + J^{\#}(\alpha_{near}) * \frac{x_{near} - x_{goal}}{\|x_{near} - x_{goal}\|} * \epsilon$$

where α is the manipulator joint configuration, x is the Cartesian coordinate, and ϵ is the step size of the RRT.

Our experiments confirm that despite the extra cost of taking a matrix inverse for each goal-directed step, the added accuracy in moving toward the goal allows the J+RRT to perform significantly better in terms of average planning times than the RRT-JT.

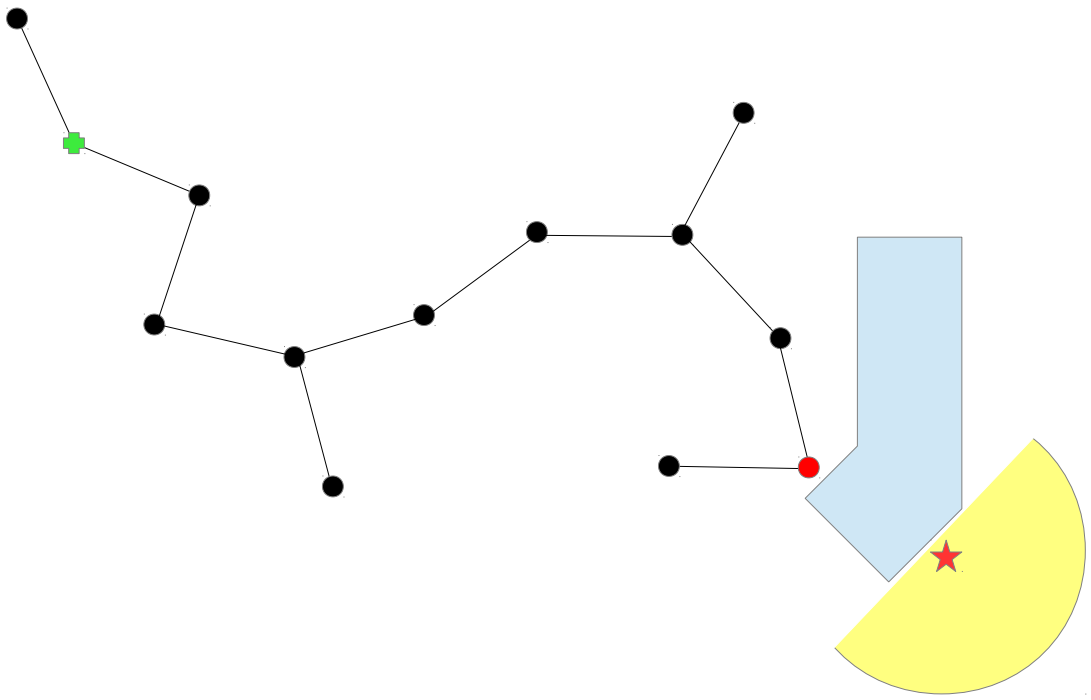


Figure 1: An uni-directional RRT attempting a bug-trap problem.

3.3 RRT and the Bug-Trap Problem

Figure 1 shows an example of a uni-directional RRT attempting to solve a problem where the goal is occluded by an object. The state of the RRT is shown after 12 iterations of the extend operation (with the probability of taking a step to goal being 35%). At this state of the tree, one can see that two problems emerge in the quest to connect the tree to the goal node:

1. Future attempts to step to goal will be taken from the red node since it is closest. All these attempts will fail because there is an obstacle in the way. In the case of the J+RRT, these steps, which constitute 35% of steps in our case, require calculating the manipulator Jacobian and its pseudo-inverse which requires matrix inversion and are therefore a significant waste of time.

2. To actually reach the goal, the RRT will have to find its way into the yellow semicircle around the goal by virtue of only random steps. Any nodes outside of this area will be either further than the red node or obstructed by the obstacle. The probability of this happening quickly is very low. Essentially, it would require that random configurations to extend toward constantly be in the bottom right corner of the workspace. For this simple example, the probability of getting such random configurations looks to be around 1/15. In a different scenario with a larger workspace and smaller step size, the probability would be much smaller.

The important takeaway is that fast RRTs tend to approach the goal in a very directed fashion but will be blocked by bug-trap type situations. Traditional RRTs rely on diffusion through random walks to go around obstacles, but this process is slow for low probability situations.

3.4 Forage RRT Implementation

The Forage RRT algorithm, described in this section, utilizes ideas from the biological foraging literature where it is shown that a multi-diffusion rate process lowers the mean time to passage for foraging problems [27, 25]. The foraging problem is the problem of finding a reward that has low probability of occurrence in a large space (with a penalty imposed on movement and searching). The foraging process model is a search process that has two diffusion rates (high and low), that result in two modes of search, or at least two distinct movement types[1] (far and near, respectively). By alternating between the two, search for an unknown goal state in a large search space is faster than traditional diffusion-based search, and covers a larger region versus greedy searches. Each movement mode is also characterized as having a specific cost:

[1]Some researchers prefer to use a single, heavy-tailed distribution to generate the two movement types [29]

low for the high diffusion rate (far mode) and high for low diffusion rate (near mode). These costs are typically associated to the attention or energy given to search versus movement.

Forage RRT replicates this idea within the context of our problem statement. The high diffusion rate mode utilizes a coarse step-size RRT algorithm, with low probability of seeking the goal, and is akin to an exploration phase. The low diffusion rate mode attempts to connect a node from the coarse tree to the goal via a fine step-size RRT algorithm with a high probability of greedily seeking the goal. The two RRTs have parameters chosen to replicate the low/high attention characteristics found in foraging. The RRT in the fine-step/high-attention mode is more guided to the goal and will also more often use the manipulator Jacobian to move. The effect of this approach for bug-trap type problems is to solve the problem presented in Section 3.3 by using the coarse step-size tree to cover joint space in search of promising approach directions to the goal, then using the fine step-size tree to avoid any small occlusions that remain on the path to the goal.

3.4.1 Goal Heap

An inefficiency in the basic RRT algorithm is that unsuccessful steps to goal may continuously be taken from the same node because that node is the closest to goal, as illustrated in Section 3.3. In RRT implementations for static environments, it is unnecessary and inefficient to continue to attempt steps to goal from nodes from which such a step has already been attempted - if it did not reach goal before, it will not again.

Implementing a goal heap is proposed as a solution to this problem. When new nodes are added to the RRT, they are also added to the goal heap along with their value (some heuristic that evaluates the fitness of the joint state). As a simple first case, the value of a node was simply the inverse of its Euclidean distance to goal

position. The heap data structure automatically orders the nodes by value so that the top of the heap is the highest value node. Once a step to goal is attempted from a node, that node is removed from the goal heap because using it again for a step to goal would be a fruitless endeavor. While the goal heap can apply to any RRT implementation, it is essential for the Forage RRT because the heap provides a node in the coarse tree for use when attempting a fine step-size J+RRT to connect to goal.

3.4.2 RRT Extend Implementation

Both coarse and fine RRT implementations use the J+RRT algorithm [33] with the proposed goal heap augmentation (J+RRT+GH). Random extensions of the search tree use the standard extension procedure of picking a random point in joint space and taking a step towards it from the nearest neighbor already in the tree. Goal-directed extensions use the Moore-Penrose pseudo-inverse Jacobian to take a step in the proper direction. The overall extend implementation is shown in Algorithm 1. Note the important step of removing a node from the goal heap once it has been tried for a step to goal.

Coarse versus fine search. The coarse search should focus more on rapidly exploring the space rather than seeking the goal. This is done by lowering the probability of stepping to goal in the coarse J+RRT+GH implementation, which also lowers the exploration cost. Since the coarse J+RRT+GH takes large step sizes, for each node extension, the link between the new node and its parent must be checked for validity by iterating through its length at an acceptable resolution and making sure each smaller step is valid. In contrast, the fine search should focus on exploring a small region with a greater intent to find the final goal state. The fine J+RRT implementation has a higher probability of stepping to goal. For the fine step-size J+RRT+GH, it is assumed that if the new node is valid, then the path connecting the parent node to the new node is also valid.

```
Data: RRT, goal, StepSize
Result: UpdatedRRT, StepResult
ρ ← random([0, 100]);
if ρ < randomExtendProbability then
    x_rand ← RANDOM_STATE();
    x_near ← NEAREST_NODE(x_rand, RRT);
    x_new ← TAKE_STEP(x_rand, x_near, StepSize);
    if isLegalPath(x_near, x_new) then
        RRT ← addVertex(x_new);
        RRT ← addEdge(x_near, x_new, StepSize);
        NodeValue ← value(x_new);
        GoalHeap ← insert(x_new, NodeValue);
        if x_new = x_randomConfig then
            return STEP_REACHED;
        else
            return STEP_PROGRESS;
        end
    else
        return STEP_COLLISION;
    end
else
    x_near ← GoalHeap → top;
    x_new ← JAC_STEP(x_near, goal, StepSize);
    if isLegalPath(x_near, x_new) then
        RRT ← addVertex(x_new);
        RRT ← addEdge(x_near, x_new, StepSize);
        if x_new = goal then
            return GOAL_REACHED;
        else
            NodeValue ← value(x_new);
            GoalHeap ← insert(x_new, NodeValue);
            GoalHeap ← remove(x_near);
            return STEP_PROGRESS;
        end
    else
        GoalHeap ← remove(x_near);
        return STEP_COLLISION;
    end
end
```

Algorithm 1: J+RRT+GH Extend Operation

3.4.3 Foraging: Alternating Between Explore and Search

The main Forage RRT algorithm is shown in Algorithm 2. The first while loop initially explores the space and builds up several promising nodes to initialize the fine searches with. The second while loop searches by attempting to connect the promising nodes from the coarse step-size tree to the goal with a fine step-size tree. Once a fine step-size RRT experiences a certain number of collisions, it is forsaken and another one is started from the next most promising node. After a given number of fine step-size searches fail, the coarse step-size tree is grown further (e.g., return to explore mode) to replenish the heap with promising start nodes. The finite search cost and restarts from alternative promising nodes allows the fine step-size RRT to be greedy, making it fast in easy situations but also robust to more challenging situations. A full list of parameters employed is given in Section 3.5.

Data: start, goal
Result: path from start to goal
$CoarseRRT \leftarrow INIT(largeStepSize, start, goal)$;
while $CoarseRRT \rightarrow size < initialSize$ **do**
  $CoarseRRT \rightarrow EXTEND()$;
end
while $result \neq GOAL\_REACHED$ **do**
  $numFailures \leftarrow 0$;
  $FineRRT \leftarrow INIT(fineStepSize, CoarseRRT \rightarrow GoalHeap \rightarrow top)$;
  while $numCollisions < maxNumCollisions$ **do**
    $FineRRT \rightarrow EXTEND()$;
  end
  increment $numFailures$;
  if $numFailures = maxNumFailures$ **then**
    for *1:percentIncrease*initialSize* **do**
      $CoarseRRT \rightarrow EXTEND()$;
    end
  end
end

Algorithm 2: Forage RRT

3.4.4 Post-Processing the Final Path

Once the fine step-size RRT has reached the goal, the found path is obtained by tracing to the root of the fine step-size tree, which is a node in the coarse step-size tree, then tracing to the root of the coarse step-size tree. Due to the mixed step-size RRTs, the raw final plan will not be attractive, especially near the start. For this purpose, path smoothing is required to make the path acceptable. This is done in a quick manner by taking pairs of nodes, attempting to connect them with a straight line, and deleting all nodes in between if successful. Empirically, it is found that picking a node from the coarse step-size plan and trying to connect it to the fine step-size plan works well. The second step is to go through remaining coarse steps and subdivide them into the desired step size to match the rest of the path. 15-20 successful smoothing steps seem to result in a satisfactorily smooth path. The computational cost for path smoothing is low compared to the path planning procedure.

3.4.5 Completeness

It is shown that the standard RRT is resolution complete because its coverage of the workspace converges to the random extension point sampling distribution over time [23]. Thus, if the sampling distribution is uniform over the workspace, the RRT will eventually cover the entire workspace which, by definition, includes the goal state. This same argument can be used to show that the Forage RRT too is complete. The exploration phase or coarse step-size RRT is designed to grow without bound until the goal is reached. Assuming the sampling distribution for random extension is uniform over the work space, the coarse RRT will eventully find the goal state.

3.5 Forage RRT Experiments and Evaluation

3.5.1 Visualization

For visualization purposes, the end-effector position of each node of the Forage RRT during a run was plotted in a 3D graph. Figure 2 shows the RRT after its initial exploration phase. It is evident at this point that nodes are sparse but cover a large volume. Figure 3 shows the Forage RRT after it has found a solution. The coarse step-size RRT has increased in size and the light blue fine step-size RRT has made a connection from the coarse node to goal.

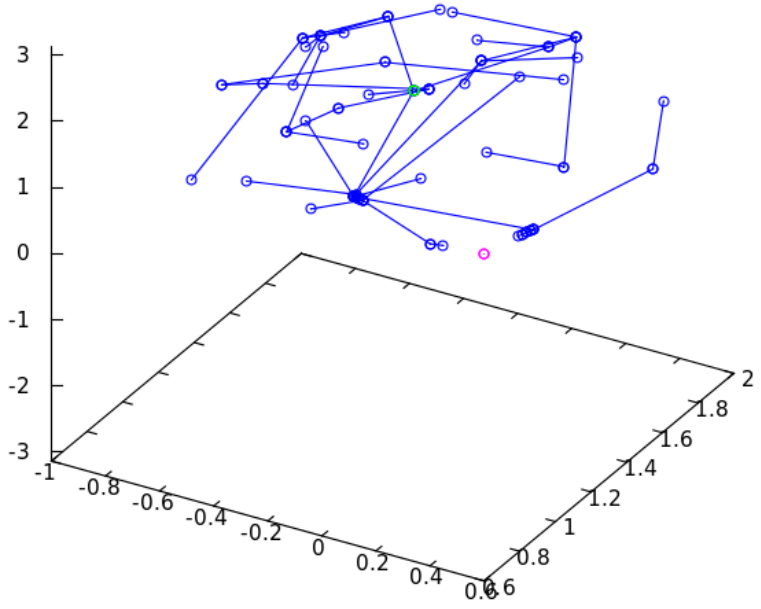


Figure 2: An exemplary coarse RRT after the initial exploration phase.

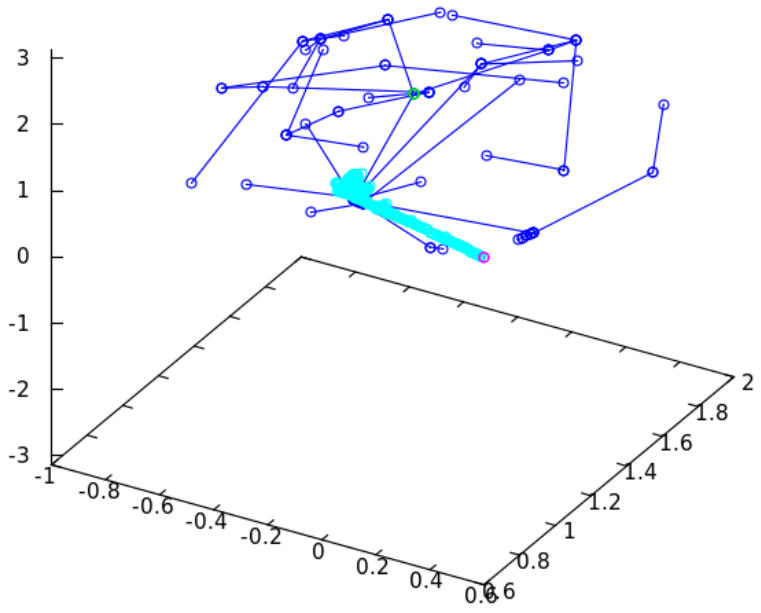


Figure 3: The Forage RRT after finding a solution.

3.5.2 Experiments

Experiments were conducted on a Schunk 7 DOF arm with the DART/GRIP simulator (developed by the GOLEMS lab at Georgia Tech) on a 2.27GHz, 8 core machine. The following algorithms were compared to the path-smoothed Forage RRT:

1. RRT-JT [34]
2. J+RRT with goal directed steps [33]
3. BiSpace RRT [12]
4. Kinematic Roadmap [1]. The paths produced from this algorithm would be too long to be used for most applications. Rather only the inverse kinematics solution, which is guaranteed to be reachable from start, would be used.

The smoothing time was included in Forage RRT results but not for the other results because the paths produced did not strictly require it. Moreover, any RRT

Table 1: Forage RRT parameters.

initialSize	50
randomExtendProbability - coarse	90
randomExtendProbability - fine	65
largeStepSize	1.3
fineStepSize	.02
maxNumCollisions	5
maxNumFailures	10
percentIncrease	.25

reaching 10,000 nodes was restarted to improve the average planning time of all planners (empirically it seems that an RRT that grows too large will have trouble connecting to the goal, so it is better to restart). The Forage RRT parameters used for testing are given in Table 1 (with common parameters for the other algorithms equaling those from the table). A couple parameters that are of particular importance in the executing time of the Forage RRT are the initialSize and the largeStepSize. The initialSize dictates the number of nodes initially included in the coarse tree to explore the space. This is discussed further in Section 4.2. The largeStepSize is the magnitude in either joint space or Cartesian space of each step taken by the coarse step-size RRT. The 1.3 was an empirically chosen number which provided good performance.

Each algorithm was tested on an easy, medium, and hard case. Figures 4- 6 depict the cases. The goal is the point where the three shown faces of the gold and blue cube meet. 50 random collision-free start configurations were computed for each case (easy, medium, and hard) and the same 50 start configurations were used for each algorithm. Each start configuration was the seed for 40 runs, thus totaling 2000 runs per algorithm per case. In the interest of time, a run was considered a failure if the RRT had to be restarted 25 times and no solution was found (such a run would have taken 400-600 seconds so it should be heavily penalized).

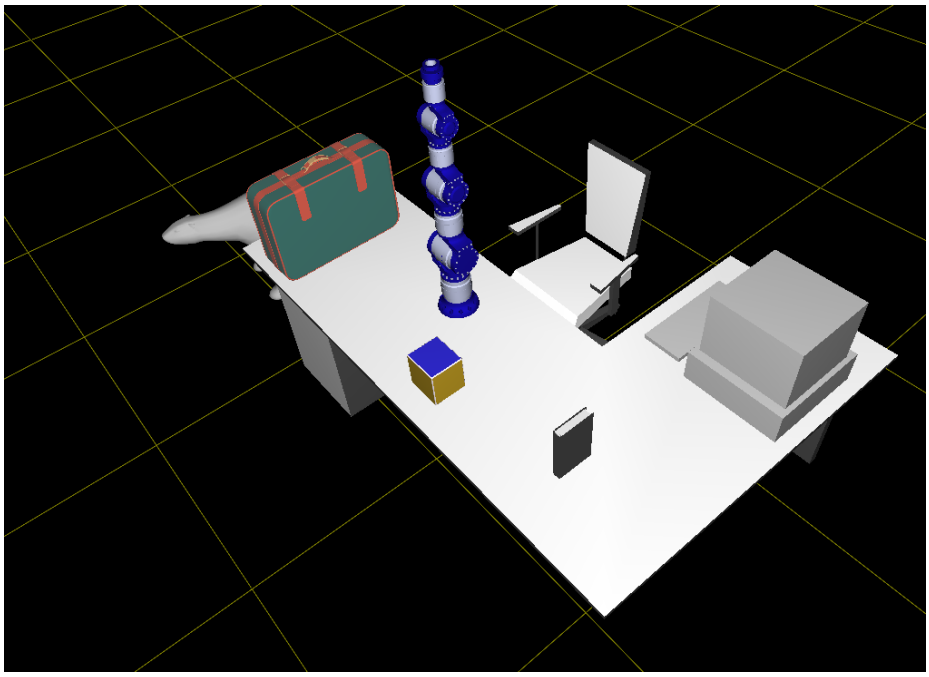


Figure 4: Easy case for testing: no obstacles between initial configuration and goal.

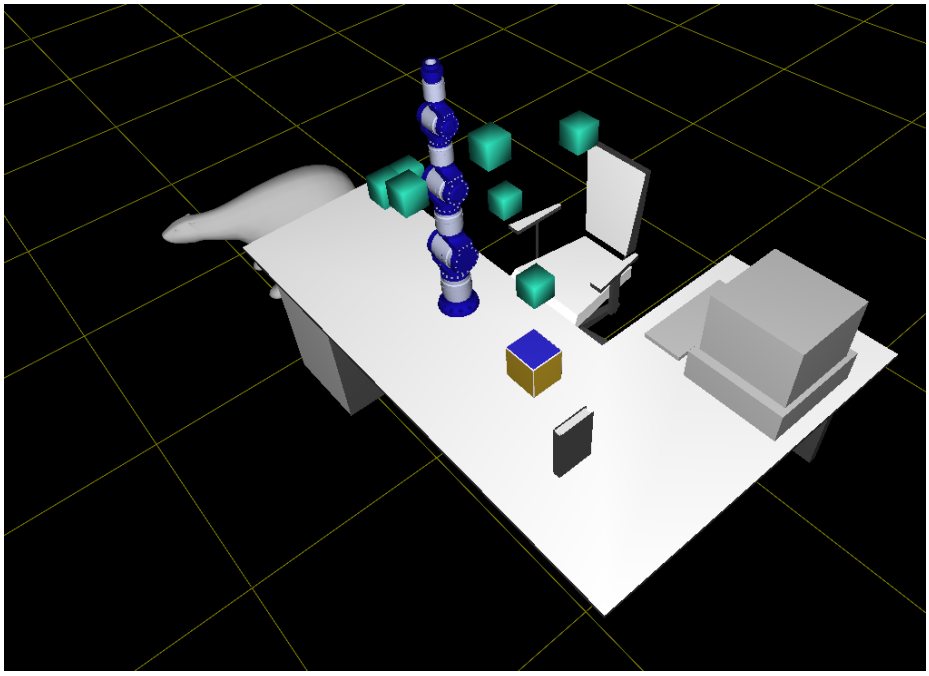


Figure 5: Medium case for testing: obstacles throughout space.

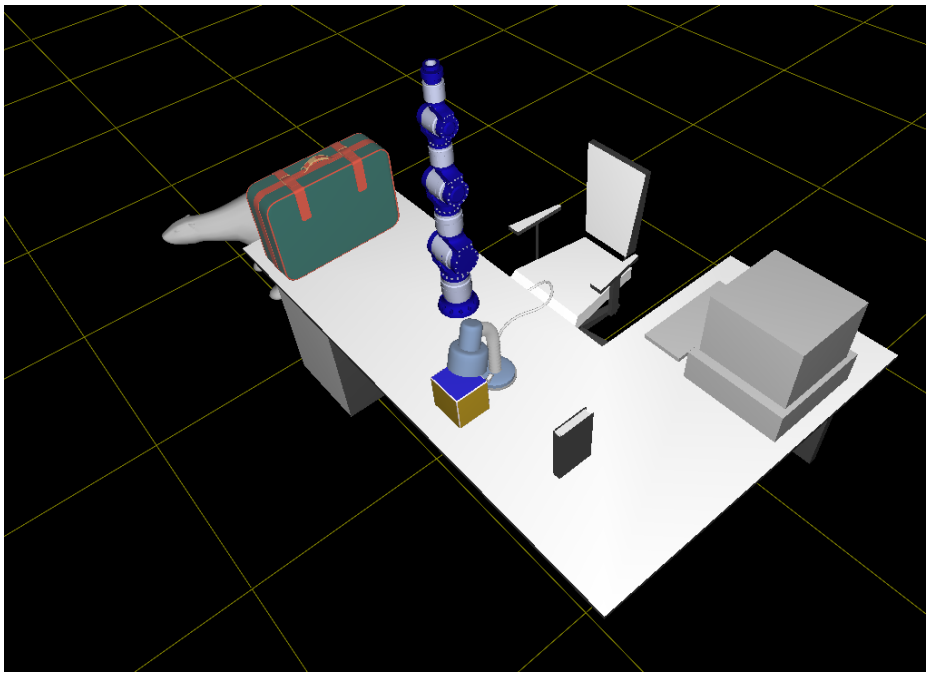


Figure 6: Hard case for testing: goal directly under obstacle, near edge of workspace.

3.5.3 Evaluation

The results of the simulation are shown in Table 2. The reported times are averaged over completed cases only. For all cases, the Forage RRT provided the lowest average completion time and also maintained a 100% completion rate for the tasks.

Table 2: Sequential experiment results. All times in seconds.

Algorithm	**Easy Avg**	**% Comp**	**Med Avg**	**% Comp**	**Hard Avg**	**% Comp**
Forage RRT	**2.92**	**100**	**3.01**	**100**	**7.52**	**100**
RRT-JT	12.12	93.7	30.51	82.7	62.62	29.0
J+RRT	4.45	96.9	21.42	92.0	65.92	53.6
BiSpace RRT	4.57	100	21.94	99.96	36.87	80.6
ACA IK	5.22	100	4.72	100	22.21	100

CHAPTER IV

FORAGE RRT IMPROVEMENTS

4.1 Parallel Implementation

4.1.1 Introduction

The non-deterministic nature of sampling based planners' performance on the same problem from run to run makes them a tempting object of parallelization. The underlying idea is that running multiple instances of a planner on the same problem decreases the planning time from the probabilistic outcome of one to the minimum of the probabilistic outcomes of many. From a performance perspective, the only potential risk is that the overhead required for managing a parallel system will outweigh the gains from taking the minimum planning time from the different instances. In reality, however, sampling based planner performance has a high enough variance that parallelization is almost always beneficial. The only downsides then are the extra complexity of implementation and the extra cost and space required for proper hardware architecture.

A parallelization approach is now presented for the Forage RRT and the results show that its implementation leads to significant improvement in average time across all 3 experiments(easy, medium, and hard).

4.1.2 Previous Work

Parallelization of sampling based motion planners, especially RRTs, is not a new idea. RRT parallelization is introduced in [7] with the strategies considered being multiple RRTs running in parallel and the multiple steps of the RRT algorithm being executed on parallel processors. Another parallelization strategy is to parallelize the most time-consuming module of the RRT, the collision checking [5]. Several RRT

parallelization strategies are explored in [11] including dividing the work of one RRT to multiple threads and a master-slave approach wherein the master thread maintains the tree and the slaves perform work which doesn't require knowledge of the tree. An interesting idea considered in [28] is how to combine single query motion planners such as the RRT with multi-query planners such as the PRM to get the benefits of both. In this case, the parallelization is of finding plans and building roadmaps simultaneously.

4.1.3 Implementation

The parallel version of the Forage RRT implements a master-slave approach with one main thread performing management and overhead roles and the others doing the brunt of the work. In some ways, the parallel Forage RRT is a hybrid strategy of the parallelization approaches mentioned above much like the Forage RRT itself is a hybridization of coarse and fine searches and a hybridization of inverse kinematics and motion planning. The outline and functions of each thread are listed as follows:

Initial Step A coarse step-size RRT is initialized and grown until its size(contained number of nodes) reaches some value greater than the number of worker threads to be implemented.

Iterate Continue until a solution is found.

Worker Threads: When seeded with an initial joint configuration, use the J+RRT to try to reach the goal of the overall planning problem. If a solution is found, set a global flag to terminate other worker threads and return the path from the seed to the goal. If a predefined number of collisions occur, return failure and wait for further instruction.

Master Thread:

1. Grow the coarse step-size RRT.

2. Seed the worker threads with initial configurations which are the current top of the goal heap of the coarse step-size RRT whenever they return failure.

Final Step Once a path to goal is returned by a worker thread, the master thread traces the path back from the worker thread's initial configuration all the way back to the global initial configuration. The reverse of this is the desired path. Simplification or other post-processing can now be done as previously.

The advantage of this approach is that very little shared memory is actually required. The only necessary information for every thread to know is whether a solution has yet been found. Otherwise, the worker threads are free to pursue their goals independently. This does require that each thread have its own copy of the world model to perform collision checking accurately, thus increasing the memory required by the planner. However, since the main goal of the Forage RRT is fast execution, the savings in planning time as a result of not needing to access shared memory is worth the memory expense.

The hybrid aspect of the parallelization approach is that all threads are technically working on different steps of the same planning problem and yet each thread is an almost completely independent planner in itself.

4.1.4 Results

The parallel implementations of the Forage RRT were tested for 10 random precomputed start configurations per case, at 40 runs per start configuration. Thus, each version was tested 400 times per case. The cases were the same as those used to test the sequential Forage RRT, shown in Figures 4- 6. Each implementation was exactly the same aside from the number of threads used in the formulation. The planning times of the parallel Forage RRT are shown in Figure 7 which plots the average planning time against the number of threads used. The 0 threads data is the data from

the sequential Forage RRT and is for comparison purposes. The experiments were conducted on the DART/GRIP simulator using an 8-core, 2.27GHz machine.

The results show that planning time improvements for the parallel implementation are substantial but sub-linear. This is to be expected because in many cases the solution is found within the first few worker threads, making additional threads inconsequential. The overall best results in the experiments come from the formulation which used 4 worker threads in addition to the master thread. The average planning times for this implementation and thus best planning times achieved with the Forage RRT are listed in Table 3.

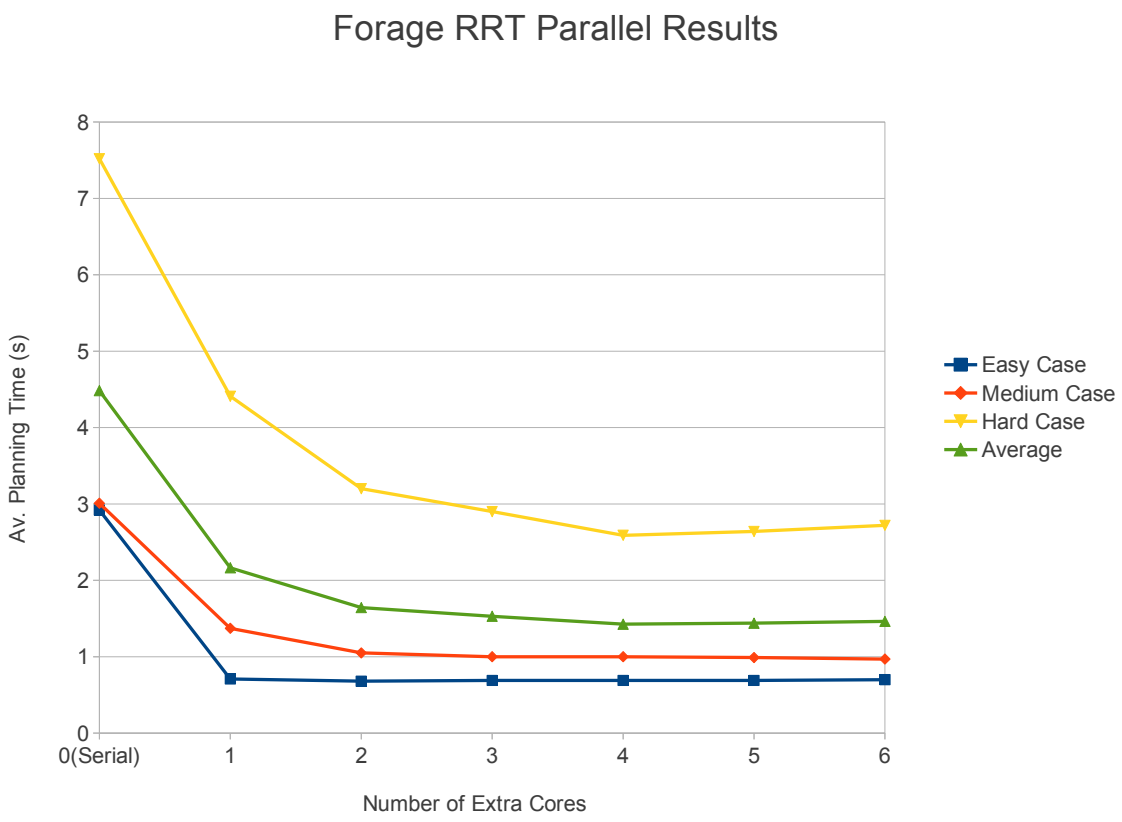


Figure 7: Parallel rrt planning time results.

Table 3: Parallel (4 core) planning results. All times in seconds.

Case:	Easy	Medium	Hard	Average
Avg. Time:	.69	1.00	2.59	**1.43**

4.2 Learning Initial Coarse RRT Size

4.2.1 Implementation

One factor which can implement the performance of the Forage RRT on a given problem is the initial size of the coarse RRT at the beginning of the algorithm. If the initial size is too small, the workspace will not be well-covered and the subsequent fine step-size RRTs will likely not be successful because they are not close enough to the goal. On the other hand, if the initial size is larger than necessary, time is wasted on building up the coarse tree when it is already large enough to produce a successful fine step-size RRT.

Since it is difficult to predict what the proper size should be ahead of time, a reasonable approach may be to automatically learn the proper initial size over time. If multiple queries are then attempted in the same workspace, later queries will have a more optimal initial coarse RRT size. A simple algorithm was implemented to test this hypothesis and it worked as follows:

1. An initial coarse RRT tree size is guessed.
2. The **final** coarse RRT tree size for each run is averaged in with the other results.
3. If the coarse RRT did not grow at all during the run, a 0 is averaged in to bring the average down.
4. After the initial run, all runs have an initial coarse RRT tree size equal to the average of the previous runs.

Two approaches to averaging were examined: averaging over all time and averaging over a window only. Theoretically, the window average is a better strategy

because it can more quickly adapt to changes in the workspace.

4.2.2 Results

To test the approach, the same methodology was used as the one for testing the sequential Forage RRT. Each strategy was implemented on 50 different precomputed random start configurations at 40 runs each for the easy and hard case. Overall, each strategy was thus tested 2000 times per case. The results are shown in Figure 8. The graph shows that the results are as expected - planning times on average are decreased with any learning at all and even more so with a learning window. The improvement is most pronounced on the easy case where the initial coarse RRT tree size could be dropped to very low numbers, essentially turning the Forage RRT into the J+RRT and harnessing the benefits of both overall.

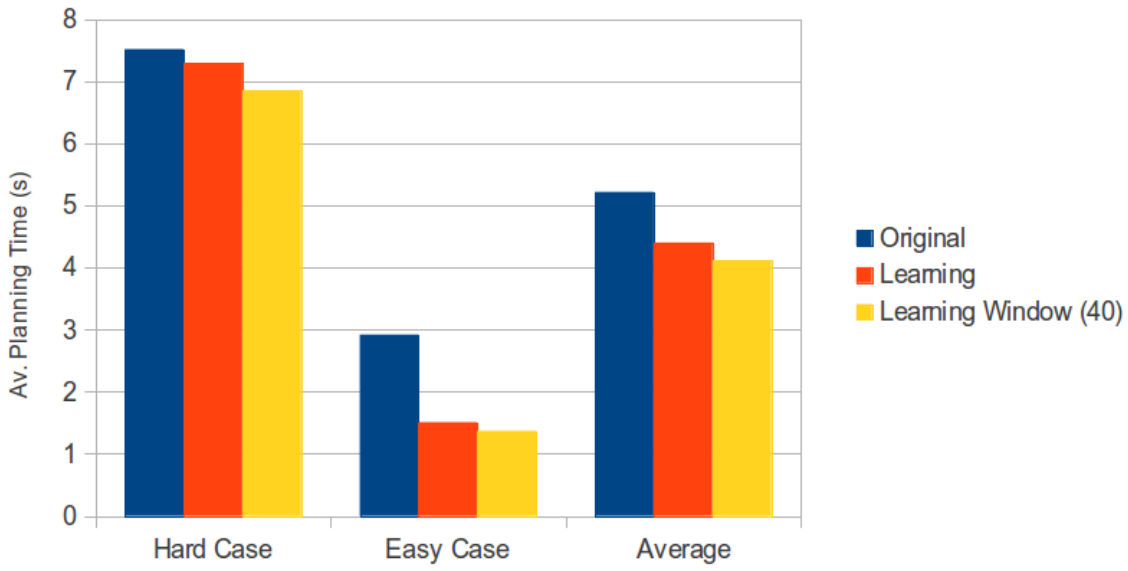


Figure 8: Results from learning approaches. All times in seconds.

CHAPTER V

OMPL ADDITION

5.1 Introduction

To facilitate the use of motion planning algorithms in industry and academia it is beneficial to have a software library which implements the most popular or most useful ones. A software library such as this allows users who need a motion planning algorithm easy access to them without requiring the time and effort required to implement them from scratch. Moreover, a motion planning library allows various planning algorithms to be benchmarked on a particular scenario quickly and easily so that the best planning algorithm can be chosen with experimental evidence rather than intuition or guessing. In motion planning, it is widely accepted that no algorithm is the best for every situation, at least as of yet. This makes benchmarking before using an important benefit of a motion planning software library.

The Open Motion Planning Library [32] undertakes the challenge of implementing a motion planning library as described above. Specifically, OMPL focuses on sampling based motion planners, making it an ideal host for the Forage RRT. This thesis implements the Forage RRT in OMPL in order to make it easily accessible to anyone who would desire to use it in the future. Challenges and implementation details of this addition are described presently.

5.2 OMPL Design Philosophy

One of the interesting paradigms in OMPL is the separation of planners and state spaces. In DART/GRIP, where the Forage RRT was originally implemented, the Forage RRT would expect a goal given in end-effector space coordinates and a start

given in joint space. By contrast, OMPL planners expect both a start and a goal given as states in the same state space. However, the state space can be any state space the user desires, so long as it performs all the functions required of a state space. The advantage here is that the same planner can be used to solve problems in many different state spaces, saving the work required to re-implement it. The disadvantage would be that different state spaces cannot be used with a given planner as required for the Forage RRT. In light of this, in order to implement the Forage RRT a new state space also had to be developed which would accommodate all requirements of a state space while also being useful for the Forage RRT.

An OMPL state space has several basic functions which planners can then use in their algorithms. The basic functions are as follows:

1. **Sampling** The state space should return a state within the state space when queried for one. Three sampling modes should be supported: uniform random, uniform near a specified point, or Gaussian near a specified point with specified Gaussian parameters.

2. **Equality** The state space should return whether two given states are equal or not.

3. **Distance** Given two states in the state space, the state space should return the distance between them.

4. **Interpolation** Given two states and a fraction between 0 and 1, the interpolation function should return a state which is between the two input states and the fraction of the distance away from the first state.

Some examples of previously implemented state space in OMPL are the Real Vector State Space(useful for joint-space planning), the SO3 state space for planning rotations, and the SE3 state space which is a compound state space made up of a real

vector component for displacement and an SO3 component for rotation. The SE3 state space is useful for planning for rigid bodies in 3D space (aka the piano movers problem).

5.3 Contribution

In order to implement the Forage RRT, which normally takes a joint space state as an initial configuration and an end-effector space state as a goal, a new state space had to be developed. The new state space is called the Manipulator State Space. Like SE3, the Manipulator State Space is a compound state space made up of a real vector component for a joint state and an SE3 component for the corresponding end-effector pose. Thus, a state in the manipulator state space should describe the state of a manipulator in terms of its joint positions, its end-effector pose, or both.

5.3.1 States

A state in the manipulator state space consists of a real vector representation of its joint position and an SE3 representation of its end-effector pose. It also contains a Boolean flag to define whether the joint configuration is known and another flag to define whether the end-effector configuration is known. If either of these is false, the values of their respective states is not to be trusted and therefore not used.

5.3.2 Manipulator State

The manipulator state space must also have at its disposal some functions that will allow it to find out some parameters of the actual manipulator being planned for. The two vital functions required are the forward kinematics of a given joint configuration and the Jacobian at a given joint configuration. These functions are essential for the distance and interpolation functions of the state space as described below.

5.3.3 Sampling

Sampling is done over the joint space component of the state space. For uniform sampling, a random vector within the bounds of the manipulator is returned as the sampled state. The returned state has a known joint configuration but an unknown end-effector pose. The end-effector pose is not calculated until necessary to save computational resources. Since the Forage RRT uses distance to goal as a measure of a node's value, all states ultimately need their end-effector poses to be defined. For other planners, however, this may not be required and is therefore not implemented as a necessity. Uniform and Gaussian sampling near given states is done much the same way with the seed of the sampling being the joint configuration of the state to sample near.

5.3.4 Equality

Two states in the manipulator state space are equal if their joint configurations are equal. A unique joint configuration uniquely identifies a manipulator state. However, a given end-effector pose could have multiple or even infinite corresponding joint states.

5.3.5 Distance

Two possibilities could occur for calculating distance between two states:

1. Both states' joint configuration is known.

2. At least one state has an unknown joint configuration.

In the first case the distance between the two states is the norm of the difference of their joint configurations, $\|j1 - j2\|$. In the second case, the distance is calculated as the SE3 distance between their end-effector configurations. In this second case, if one of the states' end-effector pose is yet unknown, it must be calculated.

5.3.6 Interpolation

Three possibilities exist for interpolation between two states:

1. Both states' joint configuration is known.

2. One state has an unknown joint configuration.

3. Both states' joint configurations are unknown.

In the first case, the output state is a simple linear interpolation between the two states with the fraction dictating how far away from the first state the output state is. That is $j_{out} = j1 + t * (j2 - j1)$ if t is the desired fraction of interpolation. In this case, the output state has a known joint configuration but an unknown end-effector pose. In the context of an RRT, this type of interpolation would generally be used to for the random growth of the tree.

In the second case, the Jacobian pseudo-inverse method is used to move the end-effector along a vector from the first state to the second.

$$j_{out} = j1 + J^{\#}(j1) * \frac{x2 - x1}{\|x2 - x1\|} * t$$

where t is the desired interpolation fraction and x is the end-effector pose of the proper state. Again, the output has known joint configuration and unknown end-effector pose. This type of interpolation is the one used for taking goal directed steps in the Forage RRT when the goal is known only as an end-effector pose.

Finally, in the case which is not commonly used for the Forage RRT, interpolation is done in SE3 space between the end-effector poses of the two states. In this case, the resultant interpolated state has only a known end-effector pose and an unknown joint configuration.

5.4 OMPL Conclusion

Both the Forage RRT and the Manipulator State Space described above have been implemented as part of OMPL. In making the Manipulator State Space, the J+RRT

is now also implemented in OMPL. This is because the RRT was implemented already and the J+RRT can be framed in an OMPL sense as simply an RRT over the Manipulator State Space.

CHAPTER VI

IK APPROACHES

6.1 Introduction

Chapter 1 discussed the shortcomings of inverse kinematics approaches in redundant manipulator planning. In these approaches, the goal is given as a Cartesian point for the end-effector to reach but before a planner is applied to the problem, this goal is converted into joint space. This translates the problem into a search over the joint space as the start is already given in these coordinates.

The advantage of this approach is mainly two-fold. First of all, the search can be done in a bidirectional fashion. That is, in the case of the RRT, a tree can be grown from both the start and the goal coordinate with the goal of each now to connect to the other. Once the trees are connected, a path can easily be traced through the parents of the connecting node on each side. Bidirectional approaches are particularly useful in situations with blockages around the start or goal because the tree starting near the blockage can more easily work its way into free space for the other tree to connect to than a single-directional tree can find a single point near an occlusion. The other advantage is that interpolation, which is often used in RRTs, is much less computationally expensive if the search is in joint space because the interpolation is the simple linear real vector interpolation instead of the pseudo-inverse Jacobian interpolation required for methods like the Forage RRT.

The disadvantage of inverse kinematics for redundant manipulators is that a closed form solution to the problem cannot generally be found because there are typically an infinite number of solutions to the problem. Thus, inverse kinematics for redundant manipulators is done with iterative numeric algorithms. The biggest issue here is

that it isn't possible to guarantee that the goal configuration could be reached from the start, for to be able to do that would mean to have the solution to the planning problem. Moreover, it is difficult to say whether the inverse kinematics solution is the best one in the sense that it is easiest to reach from start or optimal in some other meaningful sense.

The argument for the advantage of the Forage RRT, thus, is that it circumvents the problems of the inverse kinematics methods by not having an inverse kinematics solution in mind when beginning the planning but rather returning the one that was eventually reached and which can be thought of as the easiest to reach on that planner instance. This property also makes the Forage RRT resolution complete, as it will find the solution eventually given that one exists. However, an important issue arises in the usage of the Forage RRT for practical problems. Do the supposed advantages of the Forage RRT overcome its inability to implement bidirectional searches? Does the academic guarantee of resolution completeness actually manifest itself as an important advantage in practical situations over methods that do not make this claim?

In this chapter, algorithms are tested which rely on inverse kinematics on some standard benchmarking cases to attempt to get some sense of which method should be preferred for real-world applications. Ultimately, the conclusion is made that aside from some contrived low-probability situations, inverse kinematics based methods, if available, are the best approach for redundant manipulators.

6.2 MoveIt!

MoveIt [9] is a new ROS module which aims to bring together everything required to perceive, plan, and control using robots and especially manipulators. The MoveIt library contains interfaces and implementations of vision algorithms, controller interfaces for various robots, a visualizer with collision detection, and interfaces to motion planning libraries such as OMPL. MoveIt also includes functionality for benchmarking

motion planners on the same problem.

With these things in place, MoveIt is a perfect place to compare the Forage RRT to inverse kinematics based planners because it provides the infrastructure for making scenes and benchmarking as well as having access to many already-implemented planners to test against. Although the young age of MoveIt made development somewhat difficult in certain respects, the Forage RRT was able to be compared to several planners in order to get a sense of its relative performance.

Since the Forage RRT was already implemented in OMPL as discussed in Chapter 5, the easiest way to integrate it into MoveIt was to modify MoveIt's OMPL interface. This required a new state space which is very similar to the one discussed in Chapter 5 and a host of other, less interesting, modifications.

6.3 Experiments

6.3.1 Inverse Kinematics

All of the planners compared to the Forage RRT required an inverse kinematics solution before the planner could do its work. The inverse kinematics algorithm was an iterative approach which is specific to the PR2, the robot used for testing. It is worth noting that this is something of an advantage for the IK-based planners because such an algorithm would have to be worked out for each new robot individually.

6.3.2 Planners Compared

To get a sense of the relative performance of the Forage RRT, it was compared on several scenes and planning goals against the planners which are implemented by default in MoveIt through the OMPL interface. Each planner compared is now listed and a brief description of its planning strategy is given.

KPIECE [31] is a tree based planner which uses a cell decomposition to control exploration of the configuration space. The cell decomposition is particularly useful

because it allows the planner to keep track of unexplored space and expand in that direction. Because high dimensional cell decompositions are yet infeasible, this planner uses a lower dimension projection of the joint-space coordinates rather than the coordinates themselves in keeping track of occupied grid cells. In order to explore previously unexplored space efficiently, nodes on the boundary of the explored state space are given preference for expansion.

BKPIECE [31] is a bidirectional (hence the B) implementation of KPIECE. Thus, two trees are grown - one from the initial configuration and one from the goal and the trees attempt to connect.

LBKPIECE [31] is also a bidirectional KPIECE planner. However, it uses lazy collision checking [6] to improve planning time. With lazy collision checking, all links between nodes in the trees are initially considered to be collision free. Only when the path is traced are collisions actually checked for. In the event of a collision along the path, the planner continues until a new path is found.

EST [16] is similar to KPIECE in that it uses a projection of the high dimensional state space to implement a gridded search approach. The difference is that EST uses visibility and a concept of expansive spaces to guide its search to previously unexplored areas.

SBL [30] is a lazy, bidirectional version of the EST.

PRM [19], standing for Probabilistic Road Map, makes a roadmap which approximates the connectivity of the state space with connections being attempted between nodes near each other. The goal is then added to the roadmap and the planning problem becomes a graph search problem. Here, A* or similar search algorithms are implemented and can find the solution path very efficiently.

RRT [23, 24] is the basis for the Forage RRT, using a tree approach where new space is explored by picking points randomly in configuration space and taking a step towards them from their nearest neighbor already in the tree. This is a single-directional implementation of the RRT.

RRT Connect [24] Is a variant of the RRT wherein instead of taking just a step toward a randomly selected point or a goal, the tree attempts to connect all the way to the target, adding nodes to the tree all along the way. The implementation here is also bidirectional, growing a second tree from the goal.

TRRT [17] is the transition based variant of the RRT. In this version, new states are evaluated based on their cost, not just their reachability and avoidance of collisions. The main goal of this planner is paths which are low cost by some metric rather than strictly planning time.

RRT* [18] is another planner whose main purpose is not a solution in the shortest amount of time but rather a more optimal solution in a reasonable amount of time. The RRT* attempts to rewire connections constantly after finding a path until its allotted planning time runs out. The comparison against the Forage RRT was not as much for speed as whether a solution could be found in a reasonable amount of time.

6.3.3 Experimental Setup

Experiments were run using scenes and queries from the MoveIt benchmarking suite for the PR2. Because the scenes and queries were designed for the PR2 robot, it was also the one used for planning. Each experiment required the right arm of the PR2 only, which is a 7DOF redundant manipulator while the rest of the robot would remain motionless.

Each query was run from the same initial configuration 25 times. The timeout for

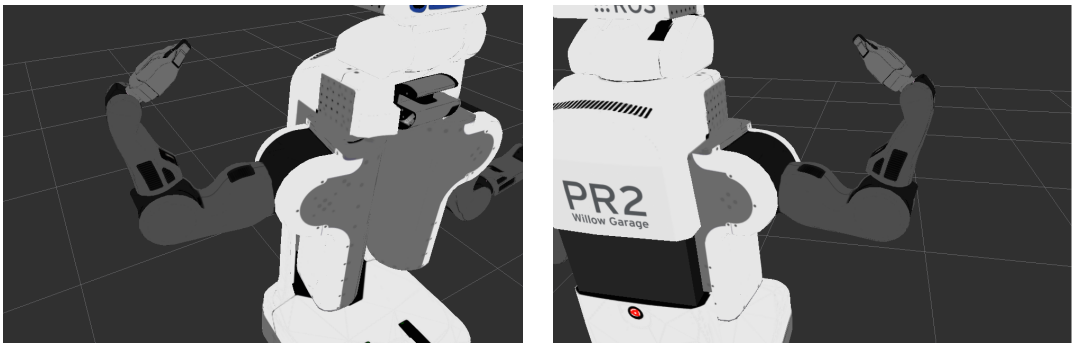
planning was 200 seconds. If no plan was found after 200s, the run was considered a failure.

Because of implementation issues in MoveIt, the Forage RRT had a goal of the x,y, and z coordinate of the query only whereas the other planners looked for the rotation aspect of the end-effector as well. This is a bit of an advantage for the Forage RRT as some of the queries may have been harder to reach when taking rotation into account. However, the results of the experiments are still a good indication of the relative performance of the Forage RRT especially given that it was quite significantly outperformed by several of the planners even with the slight advantage.

Experiments were run on an Intel i3 2.4GHz, 4GB RAM machine.

6.3.4 Experimental Manipulator

For practical reasons such as the design of benchmarking queries, the chosen manipulator for testing was the right arm of the PR2 robot included with MoveIt. It is a 7DOF arm, thus making it redundant and an ideal candidate for the Forage RRT. Figure 9 shows the PR2 right arm which was used for planning.



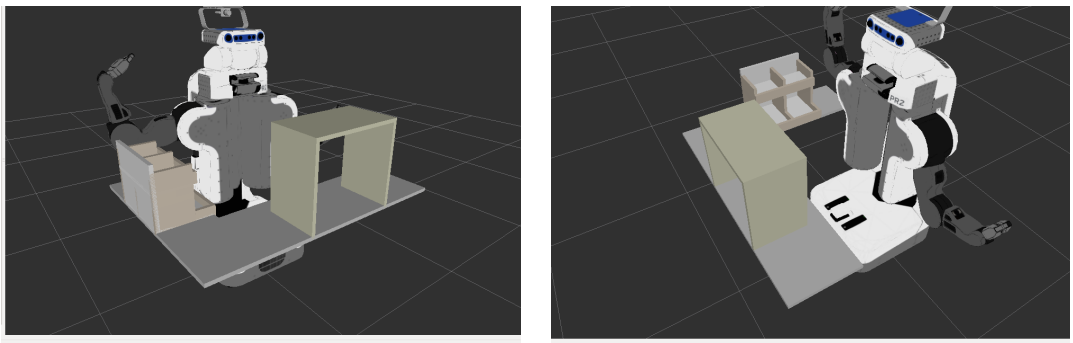
(a) Front View (b) Back View

Figure 9: The PR2 right arm manipulator used for testing.

6.3.5 Scenes

6.3.5.1 Industrial

The industrial scene is shown in Figure 10. It is one of the scenes that is included with the MoveIt benchmarking suite and is meant to replicate a factory type setting.



(a) Angle 1 (b) Angle 2

Figure 10: The industrial scene for testing.

The first query(goal specification) is shown pictorially in Figure 11. The green claw is the goal and the initial configuration is also shown in the same figure with the right arm being the motion group of interest for planning. Thus, the problem required navigating around a couple occlusions and making it into a somewhat narrow opening.

The average running times of 25 experimental runs are shown in Figure 12 with the success rate of each planner displayed in Figure 13. The results show that although the Forage RRT is somewhat competitive, several of the other planners tested would clearly be a better choice for this situation. Each member of the KPIECE family, the PRM, SBL, and RRT Connect all have significantly better average planning times as well as significantly lower variances in the planning times while maintaining the equal 100% solve rate. Note that the RRT* timing results are not important. This planner will run for the maximum allotted planning time on each run by design. The important metric for comparison in this case is the completion rate, which was around 50%. It is also interesting to note that the Forage RRT outperforms its RRT ancestor

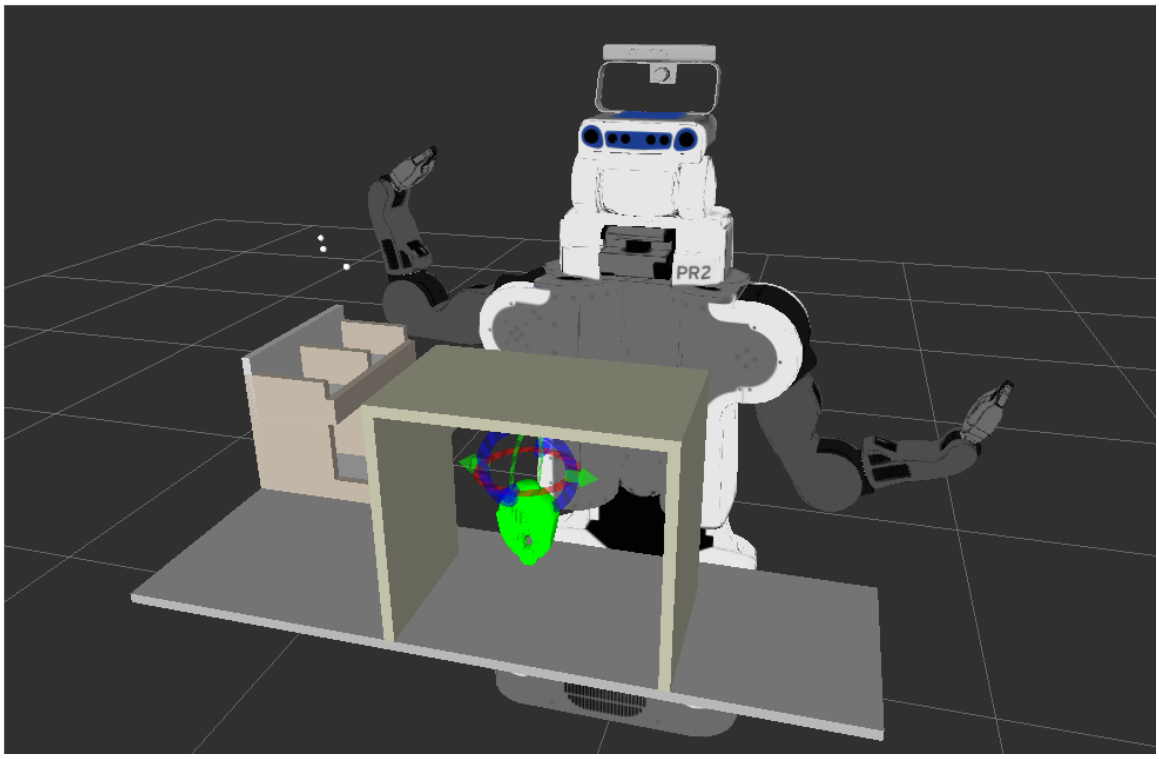


Figure 11: The first query in the industrial scene used for testing.

even though the RRT has the advantage of planning in joint space. This is further indication that the Forage RRT is the more effective approach to single direction tree search.

The second query(goal specification) is demonstrated in Figure 14. The green claw is the goal and the initial configuration is also shown in the same figure with the right arm being the motion group of interest for planning. This was something of an easier query than the first industrial specification. Only one obstacle needed to be avoided and the query was not in a narrow opening

The average running times of 25 experimental runs for query 2 are shown in Figure 15 with the success rate of each planner displayed in Figure 16. In this case, the Forage RRT had the worst average time and variances among the planners except the RRT*, which is by design. The Forage RRT did once again find a solution in under 200s each time whereas the TRRT failed to do so 50% of the time.

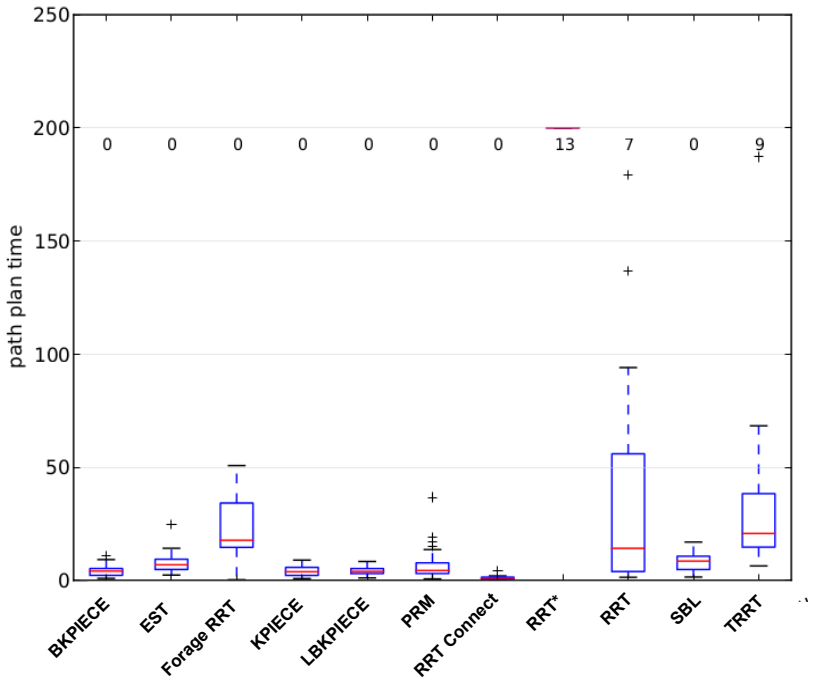


Figure 12: Results of running times for industrial query 1 over 25 runs.

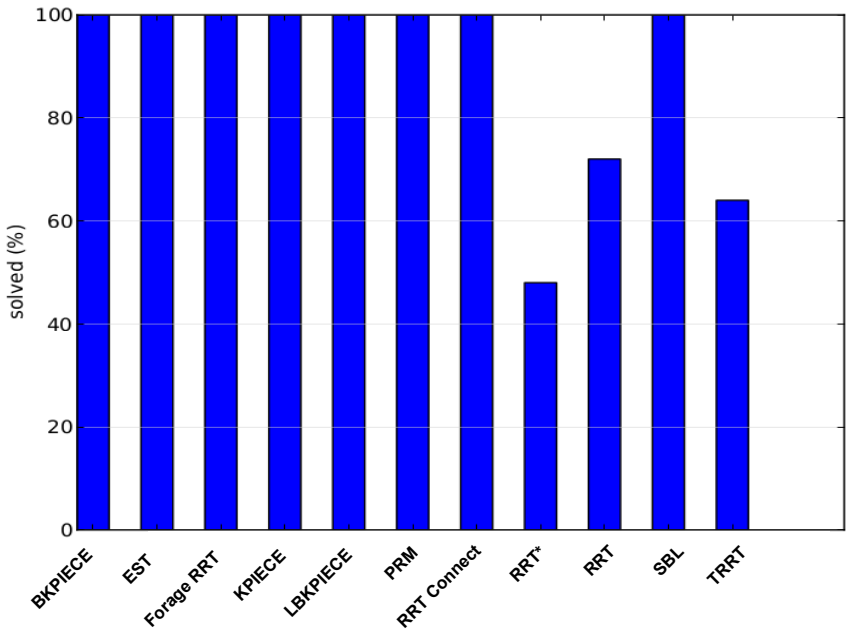


Figure 13: Results of success rates for industrial query 1 over 25 runs.

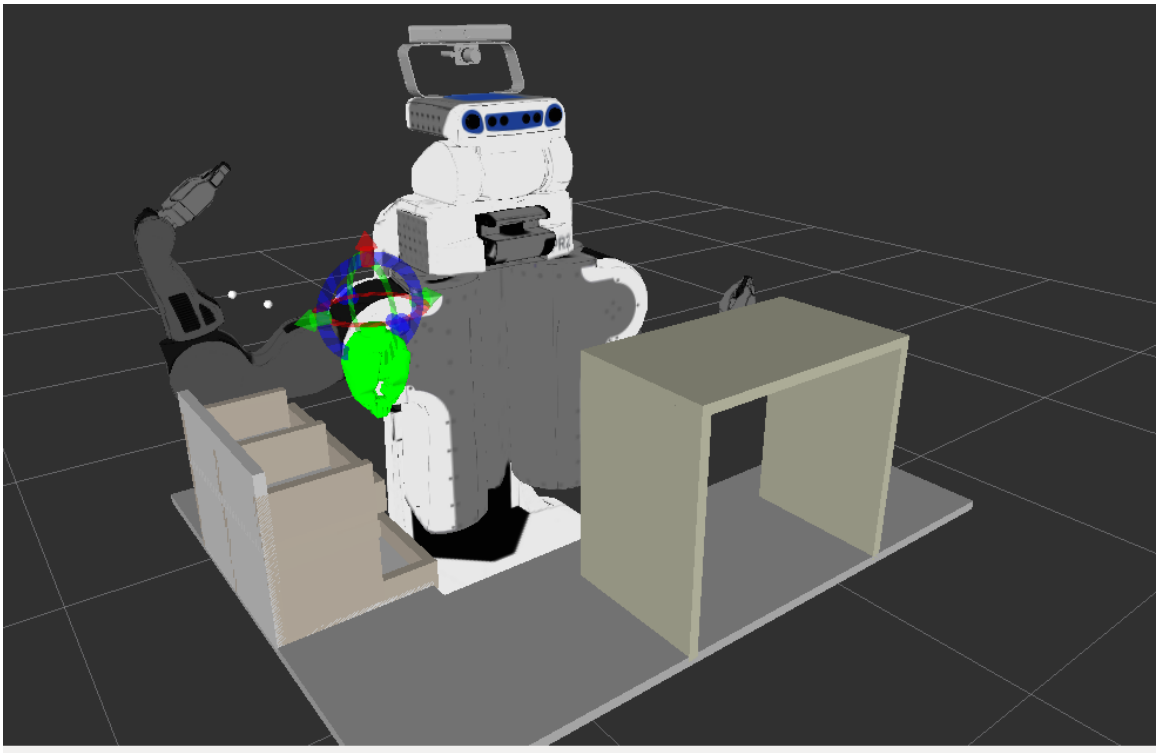


Figure 14: The second query in the industrial scene used for testing.

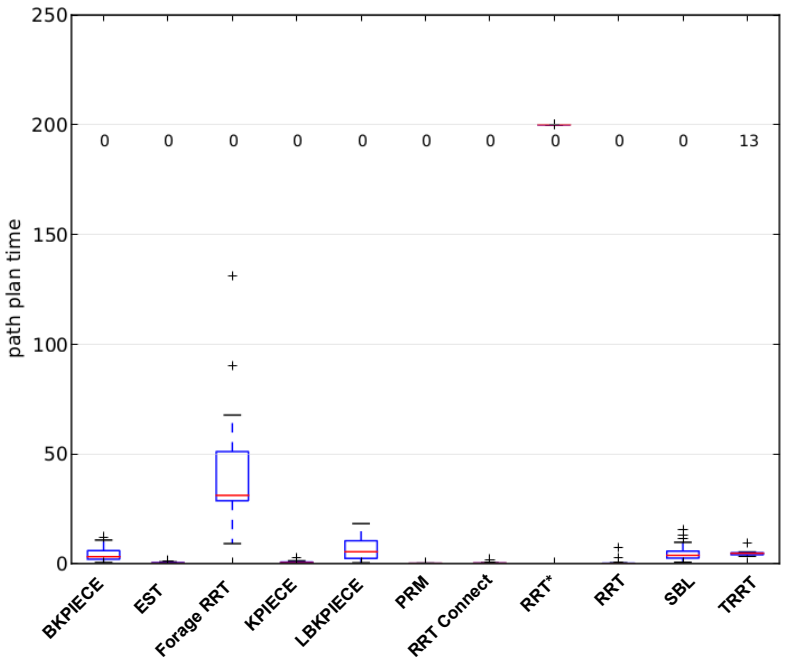


Figure 15: Results of running times for industrial query 2 over 25 runs.

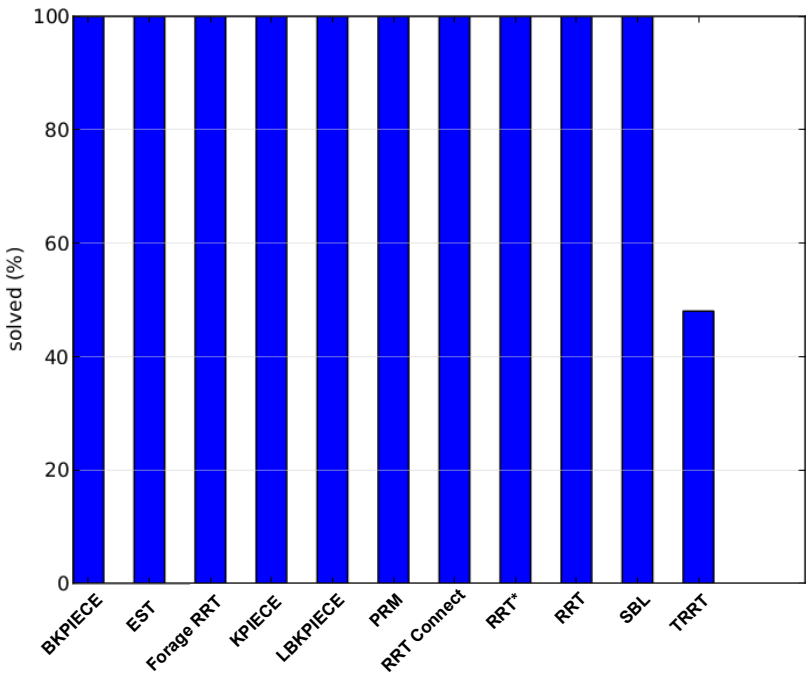
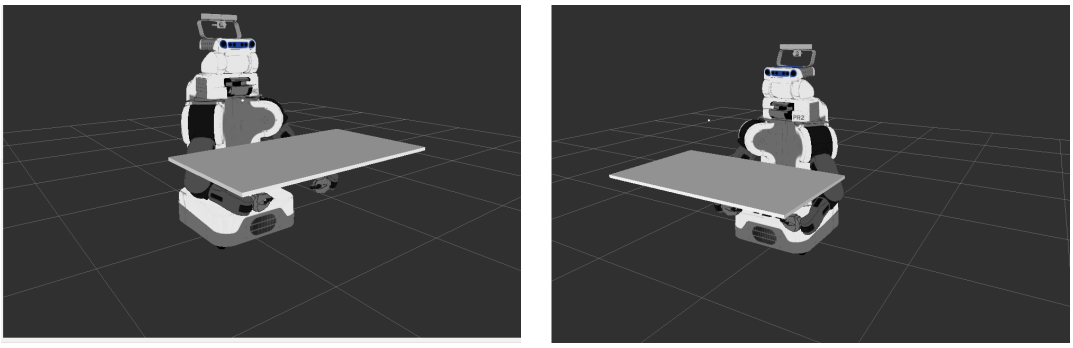


Figure 16: Results of success rates for industrial query 2 over 25 runs.

6.3.5.2 Tabletop

The tabletop scene is shown in Figure 17. It is one of the scenes that is included with the MoveIt benchmarking suite and is supposed to emulate manipulation around a table.



(a) Angle 1 (b) Angle 2

Figure 17: The tabletop scene for testing.

The first query(goal specification) is shown pictorially in Figure 18. The green

claw is the goal and the initial configuration is also shown in the same figure with the right arm being the motion group of interest for planning. This first query is quite trivial as there are no obstacles at all between the start and the goal. The only real problem is to manipulate the arm in such a way as not to violate any joint limits.

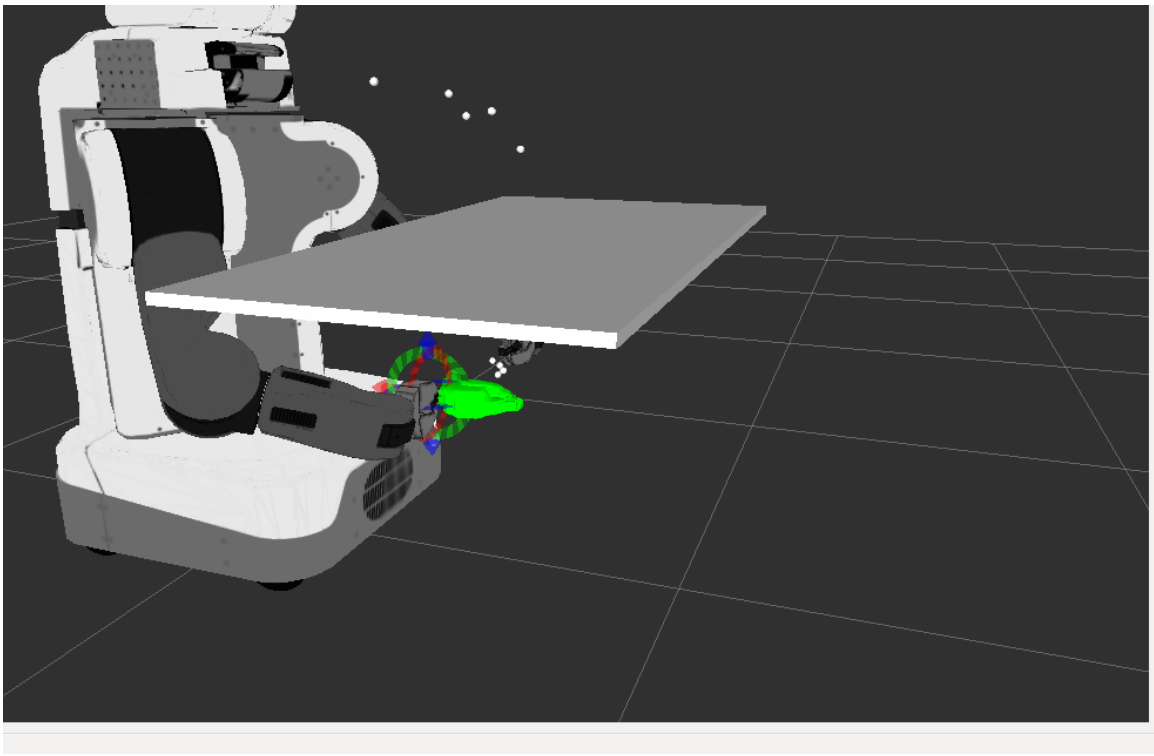


Figure 18: The first query in the tabletop scene used for testing.

The average running times of 25 experimental runs are shown in Figure 19 with the success rate of each planner displayed in Figure 20. The results show that on this trivial case, all planners produce roughly the same result of a very fast solution and a 100% completion rate. The RRT* is the only planner which did not have a comparable running time and this is because it is designed to run for the maximum time allotted to it.

The second query(goal specification) is demonstrated in Figure 21. The green claw is the goal and the initial configuration is also shown in the same figure with the right arm being the motion group of interest for planning. This query was significantly

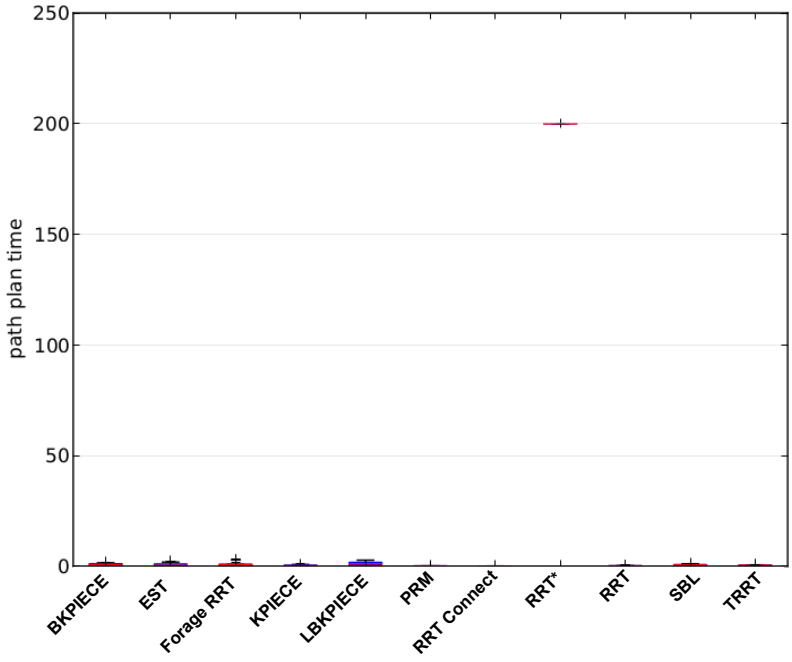


Figure 19: Results of running times for tabletop query 1 over 25 runs.

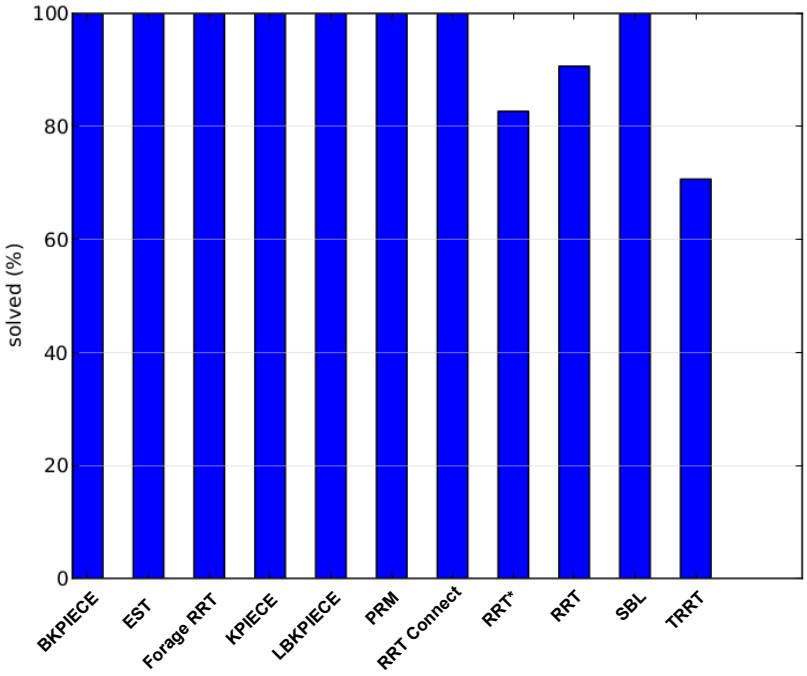


Figure 20: Results of success rates for tabletop query 1 over 25 runs.

more difficult than the first tabletop query as the path between the start and goal position was blocked in large part by the tabletop surface.

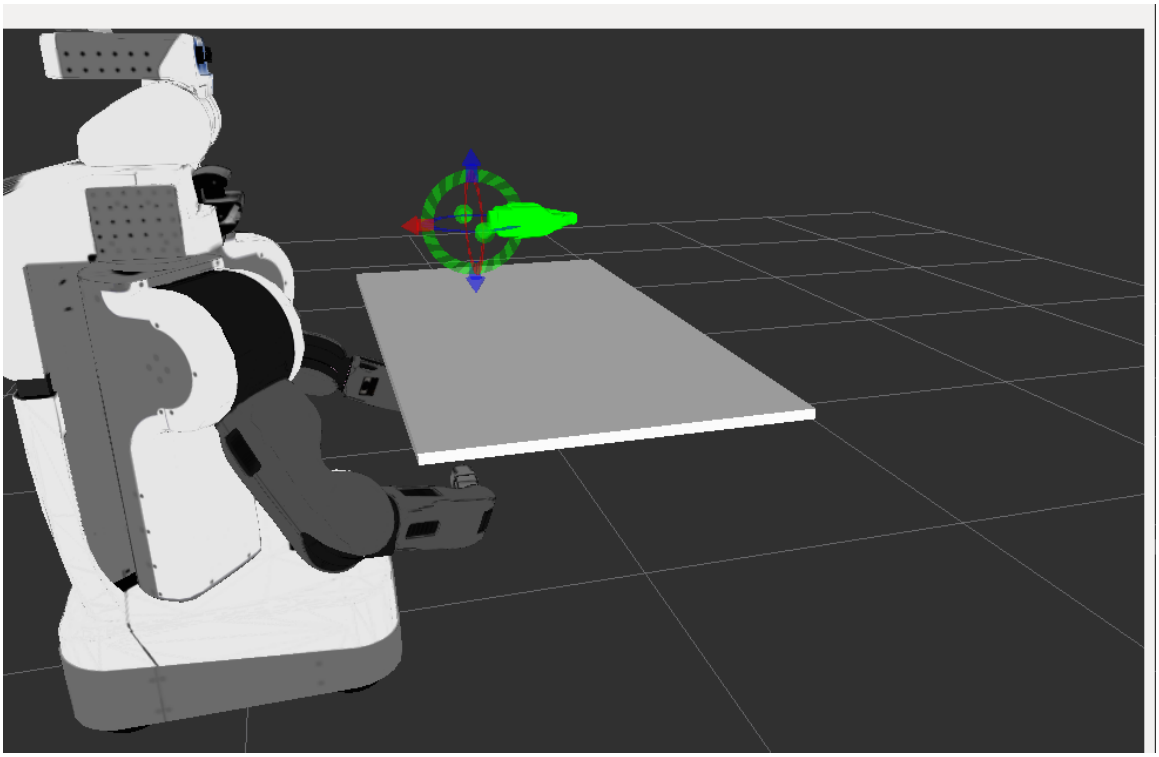


Figure 21: The second query in the tabletop scene used for testing.

The average running times of 25 experimental runs for query 2 are shown in Figure 22 with the success rate of each planner displayed in Figure 23. Interestingly, the performance of the Forage RRT was particularly disappointing in this query, producing the worst planning times of all planners by far and having a very low solve rate in general. One explanation for this is that the tabletop has a very strong negative effect on the coarse RRT's ability to explore the space to find promising nodes to attempt to connect to goal from. Essentially, the coarse RRT would have to arrive above the tabletop by random chance in order to be able to solve the query. The generally greedy nature of the fine RRT makes it ineffective in navigating around very heavy occlusions so the coarse RRT must do the exploration.

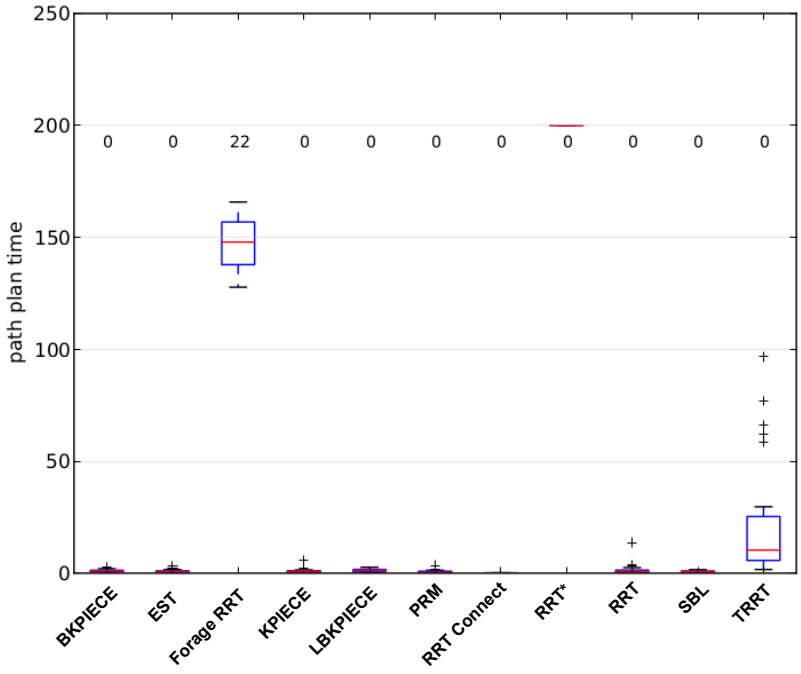


Figure 22: Results of running times for tabletop query 2 over 25 runs.

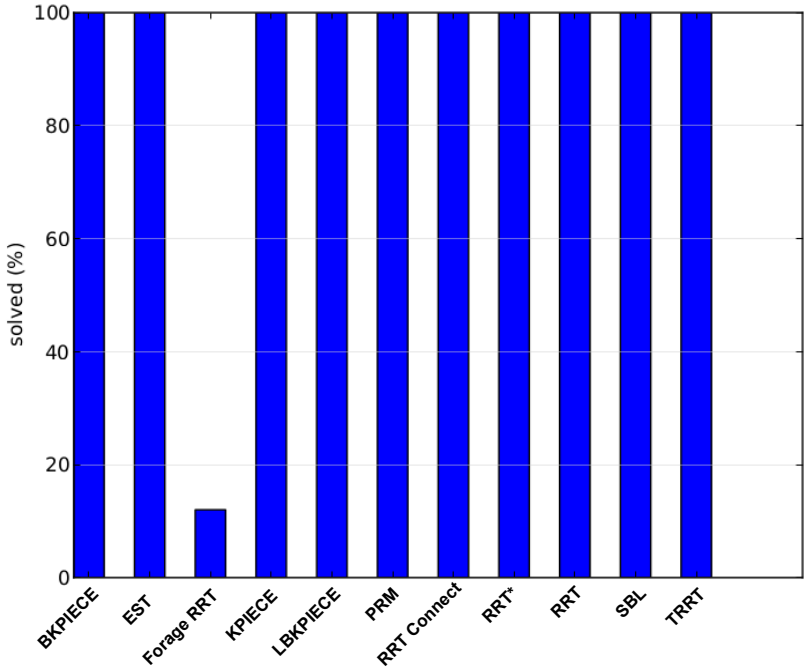
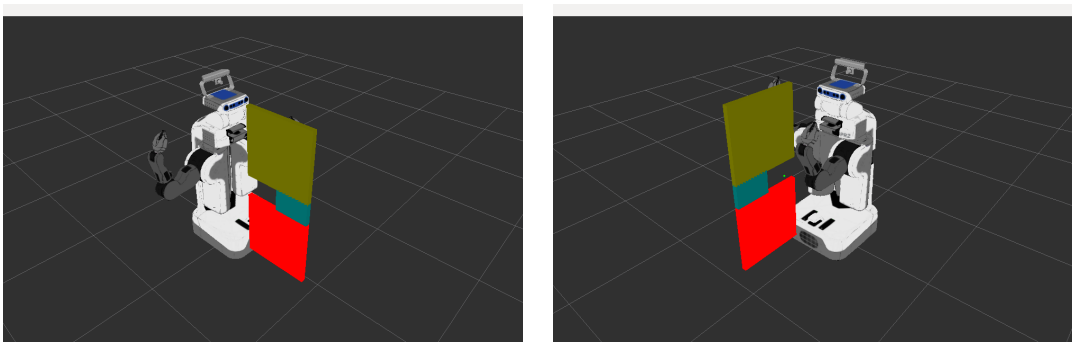


Figure 23: Results of success rates for tabletop query 2 over 25 runs.

6.3.5.3 Tunnel

The tunnel scene is shown in Figure 24. It is one of the scenes that is included with the MoveIt benchmarking suite. It models the tunnel or narrow opening problem which is typically a difficult problem for sampling based planners because the connectivity of the state space is limited by a large occlusion and the path must go through a narrow opening.



(a) Angle 1 (b) Angle 2

Figure 24: The tunnel scene for testing.

The tunnel query(goal specification) is shown pictorially in Figure 25. The green claw is the goal and the initial configuration is also shown in the same figure with the right arm being the motion group of interest for planning. This is a difficult planning problem as each planner will have to find its way through the narrow opening which connects the start configuration to the goal specification.

The average running times of 25 experimental runs are shown in Figure 26 with the success rate of each planner displayed in Figure 27. The TRRT is not shown in Figure 27 but it was one of the planners which was unable to solve the problem in under 200s even once. The planning times shown in Figure 26 are for only those planners which had at least one successful run. This is the first scene and query where some planners were never able to solve the problem. Although the Forage RRT was not able to solve the problem every time, it was more effective at solving it than EST, KPIECE, RRT*, RRT, and TRRT. Despite this positive result, it's quite clear that

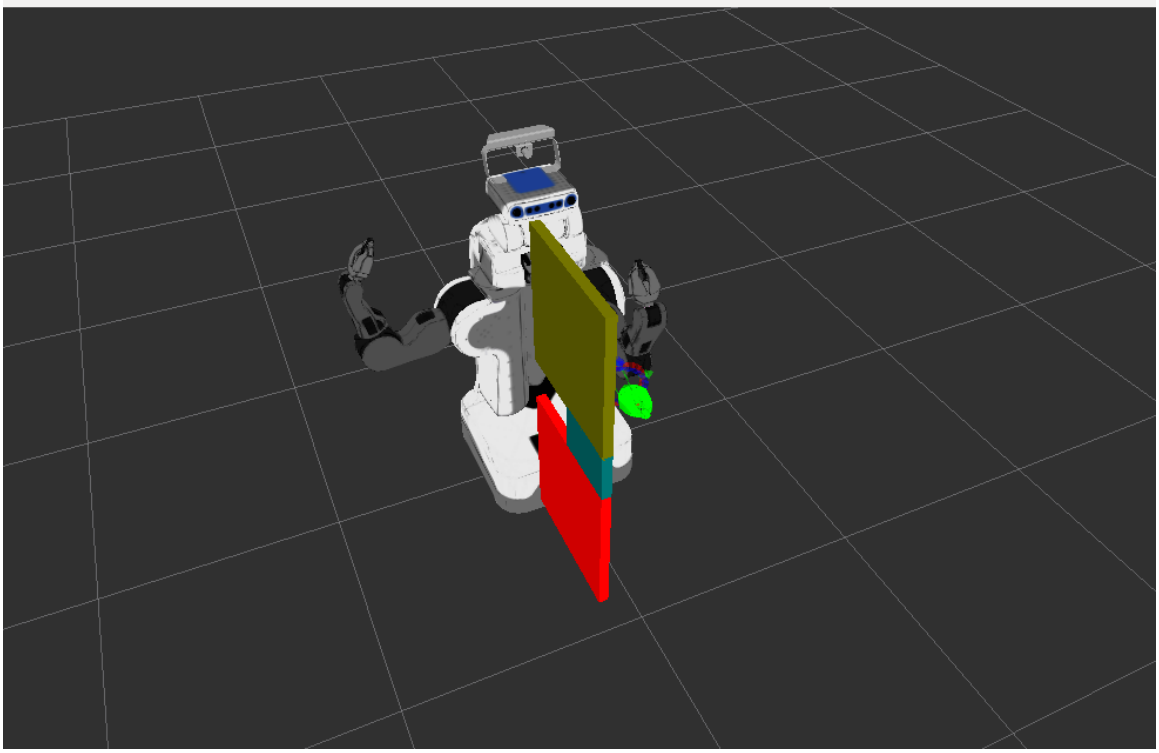


Figure 25: The query in the tunnel scene used for testing.

BKPIECE, RRT Connect, and PRM were all more effective at solving the tunnel problem in terms of success rate, average planning time, and planning time variance.

6.4 Discussion

The experiments comparing the Forage RRT to IK based planners seems to show pretty conclusively that in general, IK planners appear to be the state of the art as far as solving path planning problems for redundant manipulators in the shortest amount of time. Their ability to avoid obstacles using bidirectional approaches as well as their savings in planning time thanks for joint-space interpolation seems to heavily outweigh any extra time or uncertainty that comes with calculating inverse kinematics.

It is important to note, however, that the Forage RRT and other similar planners do have a place in the planning community. Their independence from inverse kinematics make them much easier to set up for a new robot or quick prototyping. The

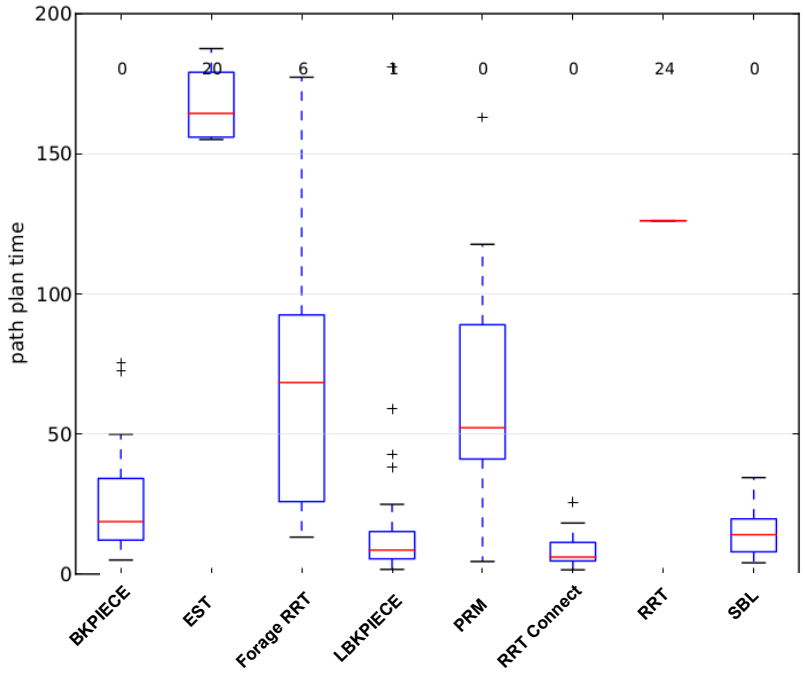


Figure 26: Results of running times for the tunnel query over 25 runs.

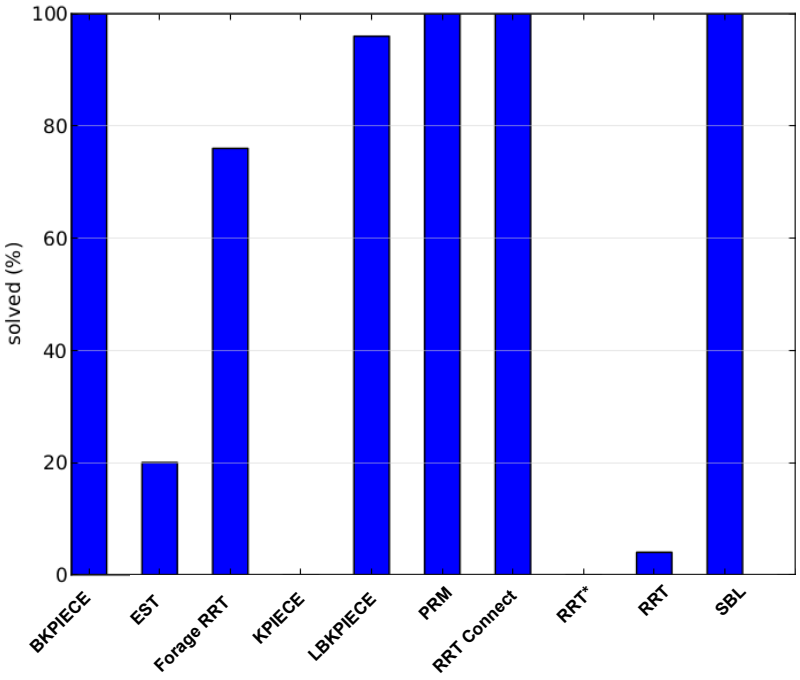


Figure 27: Results of success rates for the tunnel query over 25 runs.

Forage RRT relies solely on the Jacobian as far as robot geometry to move toward the goal. Since the Jacobian can be calculated using only robot parameters, it is significantly easier to set up than a proprietary inverse kinematics algorithm.

Altogether, however, the point remains that if possible, redundant manipulators should use inverse kinematics as the basis for their planning strategy if planning time is an important factor of performance, as it usually is.

CHAPTER VII

CONCLUSION

This thesis focused on the Forage RRT, a new motion planning algorithm for manipulators which does not require inverse kinematics. It is shown on three cases that the Forage RRT outperforms other planners which do not require inverse kinematics. However, in comparison with the top inverse-kinematics based methods, the performance of the Forage RRT leaves much to be desired.

This leaves the question: What is the value of the Forage RRT and similar methods?

The Forage RRT presents a new strategy to single-directional tree search approaches. This strategy borrows from animal foraging behavior to incorporate two diffusion rates for faster exploration of the space and an ability to get around tough obstacles without getting stuck. It is shown both intuitively and experimentally that this is a better strategy to single-directional searching than the classic RRT strategy. Ultimately, this strategy appears to be the most significant contribution of the Forage RRT and this thesis. If, for whatever reason, a bidirectional approach cannot be implemented, the foraging strategy is a good way to get reasonable results from searching or planning in a cluttered environment.

Moreover, the structure of the Forage RRT indicates some room for improvements in the future. Aside from quickly exploring the search space, the coarse step-size RRT provides a frame or lattice which can be used beneficially for replanning. The coarse RRT frame provides a quick connection to the initial configuration from many areas in the search space. This means that if a new query were introduced in the same environment, the coarse RRT could be reused. Other replanning strategies where the

environment changes slightly may be considered as well. The value of a node as far as attempting to go to goal in this thesis was simply the inverse of its distance to goal. Quite conceivably, better value measures such as visibility or difference away from other coarse nodes may exist.

In general, the Forage RRT framework is quite flexible and general and should allow for useful algorithms to stem from it in the future. For now, however, there are better planners for redundant manipulators and manipulators in general.

APPENDIX A

DEFINITIONS

Bidirectional: Approaches which try not only to find a path from start to goal but also goal to start because the reverse of this path is also a valid planning solution.

Bug-trap problem: Motion planning problem wherein either the goal or start configuration is largely occluded such that entering or exiting is difficult.

Completeness: Property of planning algorithm meaning the algorithm finds a path if one exists and returns failure if one does not exist.

Connectivity: A measure of how cluttered an environment is i.e. how easily one could find a path from one area to another.

End-effector configuration: A description of the state of the end-effector. Typically a position and a rotation relative to some frame.

End-effector pose: The translation and rotation of the end-effector relative to a known frame.

Forward kinematics: The problem of solving for a manipulator's end-effector pose given its joint state.

Inverse kinematics: The problem of solving for a manipulator's joint state given its end-effector pose.

Jacobian: A matrix representing the relationship between a manipulator end effector's velocity in 3D space and the angular velocity of its joints.

Joint configuration: The states of a manipulator's joints i.e. the angle of each joint relative to the previous link.

Joint state: The state of a manipulator's joints i.e. the angle of each joint relative to the previous link.

Joint space: The set of all possible joint states for a given manipulator. Could be infinite or finite.

Lazy: Approaches which initially assume all connections between nodes are valid. Only when the final path is computed are the links checked. Ideally, minimizes computational expense of unnecessary collision checking.

Node: An individual element in a graph or tree structure. Nodes are connected by links or branches.

Query: A planning problem, usually stated for manipulators in terms of an end-effector pose.

Resolution Completeness: Property of algorithm meaning the algorithm finds a path if one exists but does not return failure if one does not. Instead, it would run forever.

SE3: A space describing rigid body transformations as translations and rotations in 3D space.

SO3: A space describing rigid body rotations in 3D space.

State Space: The set of all possible states of a system. Could be infinite or finite.

REFERENCES

[1] Ahuactzin, J. and Gupta, K., "The kinematic roadmap: A motion planning based global approach for inverse kinematics of redundant robots," *IEEE Transactions on Robotics and Automation*, vol. 15, no. 4, pp. 653–669, 1999.

[2] Amato, N. M. and Wu, Y., "A randomized roadmap method for path and manipulation planning," in *IEEE International Conference on Robotics and Automation*, vol. 1, pp. 113–120, 1996.

[3] Bertram, D., Kuffner, J., Dillmann, R., and Asfour, T., "An integrated approach to inverse kinematics and path planning for redundant manipulators," in *IEEE International Conference on Robotics and Automation*, pp. 1874–1879, 2006.

[4] Bessiere, P., Ahuactzin, J., and Talbi, E.G.and Mazer, E., "The Ariadne's clew algorithm: global planning with local methods," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, vol. 2, pp. 1373–1380, 1993.

[5] Bialkowski, J., Karaman, S., and Frazzoli, E., "Massively parallelizing the rrt and the rrt," in *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*, pp. 3513–3518, IEEE, 2011.

[6] Bohlin, R. and Kavraki, E., "Path planning using lazy prm," in *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, vol. 1, pp. 521–528, IEEE, 2000.

[7] Carpin, S. and Pagello, E., "On parallel rrts for multi-robot systems," in *Proc. 8th Conf. Italian Association for Artificial Intelligence*, pp. 834–841, 2002.

[8] Chang, P., "A closed-form solution for inverse kinematics of robot manipulators with redundancy," *IEEE Journal of Robotics and Automation*, vol. 3, no. 5, pp. 393–403, 1987.

[9] Chitta, S., Sucan, I., and Cousins, S., "Moveit!," *IEEE Robotics Automation Magazine*, vol. 19, no. 1, pp. 18–19, 2012.

[10] Connolly, C., Burns, J., and Weiss, R., "Path planning using Laplace's equation," in *IEEE International Conference on Robotics and Automation*, pp. 2102–2106, 1990.

[11] Devaurs, D., Siméon, T., and Cortés, J., "Parallelizing rrt on distributed-memory architectures," 2011.

[12] Diankov, R., Ratliff, N., Ferguson, D., Srinivasa, S., and Kuffner, J., "Bispace planning: Concurrent multi-space exploration," in *Proceedings of Robotics: Science and Systems*, vol. 63, 2008.

[13] Ge, S. and Cui, Y., "New potential functions for mobile robot path planning," *IEEE Transactions on Robotics and Automation*, vol. 16, no. 5, pp. 615–620, 2000.

[14] Goldenberg, A., Benhabib, B., and Fenton, R., "A complete generalized solution to the inverse kinematics of robots," *IEEE Journal of Robotics and Automation*, vol. 1, no. 1, pp. 14–20, 1985.

[15] Guez, A. and Ahmad, Z., "Solution to the inverse kinematics problem in robotics by neural networks," in *IEEE International Conference on Neural Networks*, pp. 617–624, 1988.

[16] Hsu, D., Latombe, J.-C., and Motwani, R., "Path planning in expansive configuration spaces," in *Robotics and Automation, 1997. Proceedings., 1997 IEEE International Conference on*, vol. 3, pp. 2719–2726, IEEE, 1997.

[17] Jaillet, L., Cortés, J., and Siméon, T., "Sampling-based path planning on configuration-space costmaps," *Robotics, IEEE Transactions on*, vol. 26, no. 4, pp. 635–646, 2010.

[18] Karaman, S. and Frazzoli, E., "Sampling-based algorithms for optimal motion planning," *The International Journal of Robotics Research*, vol. 30, no. 7, pp. 846–894, 2011.

[19] Kavraki, L. E., Svestka, P., Latombe, J.-C., and Overmars, M. H., "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *Robotics and Automation, IEEE Transactions on*, vol. 12, no. 4, pp. 566–580, 1996.

[20] Khatib, O., "Real-time obstacle avoidance for manipulators and mobile robots," *International Journal of Robotics Research*, vol. 5, no. 1, pp. 90–98, 1986.

[21] Klein, C. A. and Huang, C.-H., "Review of pseudoinverse control for use with kinematically redundant manipulators," *Systems, Man and Cybernetics, IEEE Transactions on*, no. 2, pp. 245–250, 1983.

[22] Koren, Y. and Borenstein, J., "Potential field methods and their inherent limitations for mobile robot navigation," in *IEEE International Conference on Robotics and Automation*, pp. 1398–1404, 1991.

[23] LaValle, S., "Rapidly-exploring random trees a new tool for path planning," 1998.

[24] LaValle, S. and Kuffner Jr., J., "Rapidly-exploring random trees: Progress and prospects," 2000.

[25] Moreau, M., Bénichou, O., Loverdo, C., and Voituriez, R., "Chance and strategy in search processes," *J. Stat. Mech.*, p. P12006, 2009.

[26] Parker, J., Khoogar, A., and Goldberg, D., "Inverse kinematics of redundant robots using genetic algorithms," in *IEEE International Conference on Robotics and Automation*, pp. 271–276, 1989.

[27] Passino, K. M., *Biomimicry for Optimization, Control, and Automation*. Springer, 2005.

[28] Plaku, E., Bekris, K. E., Chen, B. Y., Ladd, A. M., and Kavraki, L. E., "Sampling-based roadmap of trees for parallel motion planning," *Robotics, IEEE Transactions on*, vol. 21, no. 4, pp. 597–608, 2005.

[29] Plank, M. and James, A., "Optimal foraging: Lévy pattern or process?," *J. R. Soc. Interface*, vol. 5, no. 26, pp. 1077–1086, 2008.

[30] Sánchez, G. and Latombe, J.-C., "A single-query bi-directional probabilistic roadmap planner with lazy collision checking," in *Robotics Research*, pp. 403–417, Springer, 2003.

[31] Şucan, I. A. and Kavraki, L. E., "Kinodynamic motion planning by interior-exterior cell exploration," in *Algorithmic Foundation of Robotics VIII*, pp. 449–464, Springer, 2009.

[32] Sucan, I. A., Moll, M., and Kavraki, E., "The open motion planning library," *Robotics & Automation Magazine, IEEE*, vol. 19, no. 4, pp. 72–82, 2012.

[33] Vahrenkamp, N., Berenson, D., Asfour, T., Kuffner, J., and Dillmann, R., "Humanoid motion planning for dual-arm manipulation and regrasping tasks," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 2464–2470, 2009.

[34] Vande Weghe, M., Ferguson, D., and Srinivasa, S., "Randomized path planning for redundant manipulators without inverse kinematics," in *IEEE-RAS International Conferenceon Humanoid Robots*, pp. 477–482, 2007.

[35] Yao, Z. and Gupta, K., "Path planning with general end-effector constraints: Using task space to guide configuration space search," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 1875–1880, 2005.

VITA

Leo Keselman was born in Semiluki, Russia in 1989 and moved to the Atlanta area in 1997. He graduated from Wheeler High School in 2007, Georgia Tech ECE B.S. in 2012 and Georgia Tech ECE M.S. in 2014. He is a big baseball fan and also enjoys canoeing, playing with his dogs, movies, music, video games, and much much more!

As of the time of this writing, he is working as a Software Product Development Engineer for Fanuc America Corporation in Rochester Hills, MI.